ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΚΑΤΑΣΚΕΥΗ ΑΥΤΟΝΟΜΟΥ ΟΧΗΜΑΤΟΣ ΚΑΙ ΑΝΙΧΝΕΥΣΗ ΛΩΡΙΔΑΣ ΜΕ ΜΕΘΟΔΟΥΣ ΜΗΧΑΝΙΚΗΣ ΟΡΑΣΗΣ

Διπλωματική Εργασία

Μιχαήλ Μόκκας

Επιβλέπουσα: Παναγιώτα Τσομπανοπούλου

Φεβρουάριος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΚΑΤΑΣΚΕΥΗ ΑΥΤΟΝΟΜΟΥ ΟΧΗΜΑΤΟΣ ΚΑΙ ΑΝΙΧΝΕΥΣΗ ΛΩΡΙΔΑΣ ΜΕ ΜΕΘΟΔΟΥΣ ΜΗΧΑΝΙΚΗΣ ΟΡΑΣΗΣ

Διπλωματική Εργασία

Μιχαήλ Μόκκας

Επιβλέπουσα: Παναγιώτα Τσομπανοπούλου

Φεβρουάριος 2022

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CONSTRUCTION OF AUTONOMOUS VEHICLE AND LANE DETECTION WITH COMPUTER VISION METHODS

Diploma Thesis

Michail Mokkas

Supervisor: Panagiota Tsompanopoulou

February 2022

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπουσα **Παναγιώτα Τσομπανοπούλου**
Αναπληρώτρια Καθηγήτρια, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Μιχαήλ Βασιλακόπουλος**
Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Αθανάσιος Φεύγας**
Μέλος ΕΔΙΠ, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές ευχαριστίες μου στην Δρ. Τσομπανοπούλου Παναγιώτα, επιβλέπουσα της παρούσας εργασίας για την πολύτιμη βοήθεια και καθοδήγησή της καθ' όλη την διάρκεια της συγγραφής.

Στη συνέχεια, θα ήθελα να ευχαριστήσω τον φίλο και συνάδελφο Ζέρβα Χρήστο για την άψογη συνεργασία μας κατά την μελέτη και κατασκευή του οχήματος και του ρομποτικού βραχίωνα.

Τέλος, θα ήθελα να εκφράσω την ευγνωμοσύνη μου και να αφιερώσω την διπλωματική μου εργασία στην οικογένεια και τους αγαπημένους μου που με στήριξαν όλα αυτά τα χρόνια.

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

«Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής».

Ο Δηλών

Μιχαήλ Μόκκας

Διπλωματική Εργασία

ΚΑΤΑΣΚΕΥΗ ΑΥΤΟΝΟΜΟΥ ΟΧΗΜΑΤΟΣ ΚΑΙ ΑΝΙΧΝΕΥΣΗ ΛΩΡΙΔΑΣ ΜΕ ΜΕΘΟΔΟΥΣ ΜΗΧΑΝΙΚΗΣ ΟΡΑΣΗΣ

Μιχαήλ Μόκκας

Περίληψη

Η σύγχρονη εποχή χαρακτηρίζεται από ραγδαία και συνεχή τεχνολογική ανάπτυξη στην οποία κυριαρχεί ο ανταγωνισμός. Ο χρόνος και οι έγκαιρες και ταχείες αποφάσεις είναι καθοριστικής σημασίας για την εμφάνιση νέων τεχνολογικών καινοτομιών. Η υιοθέτηση τέτοιων πρωτοπόρων τεχνολογιών, που επιδιώκονται σε ένα σύγχρονο περιβάλλον, όπως της εποχής μας, έχει στόχο να ξεπεράσει κάθε προηγούμενη προσπάθεια, ώστε να επικρατήσει σε οικονομικό και τεχνολογικό επίπεδο.

Η παρούσα διπλωματική εργασία επικεντρώνεται στην εξέλιξη της τεχνολογίας στα μέσα μεταφοράς και ειδικότερα στην αυτόνομη οδήγηση. Στα πλαίσια αυτής της εργασίας κατασκευάστηκε ένα όχημα με την προσθήκη ενός ρομποτικού βραχίωνα. Με την ανάπτυξη ενός αλγορίθμου αναγνώρισης λωρίδας έγινε εφικτή η αυτόνομη οδήγηση του οχήματος σε μια λωρίδα κίνησης οριοθετημένης διαδρομής σε πραγματικό χρόνο. Ο αλγόριθμος βασίστηκε σε τεχνικές παλαιότερων ερευνών που δοκιμάστηκαν ως επί το πλείστον σε βιντεοσκοπημένες και όχι πραγματικές διαδρομές. Η παρούσα μελέτη αποτελεί έναυσμα για μελλοντικές εφαρμογές και καινοτομίες που περιλαμβάνουν και την χρήση του ρομποτικού βραχίωνα.

Λέξεις-κλειδιά:

Αυτόνομη Οδήγηση, Αυτόνομο Όχημα, Ρομπότ, Ρομποτικός Βραχίωνας, Αναγνώριση Λωρίδας Πραγματικού Χρόνου, Αλγόριθμος Αναγνώρισης Λωρίδας, OpenCV, Gaussian Blur, Canny Edge Detection, Hough Transform.

Diploma Thesis

CONSTRUCTION OF AUTONOMOUS VEHICLE AND LANE DETECTION WITH COMPUTER VISION METHODS

Michail Mokkas

Abstract

Our modern era is characterized by rapid and continuous technological development which is dominated by competition. Time, timely and rapid decisions are of decisive importance for the emergence of new technological innovations. The adoption of such pioneering technologies aims to surpass any previous effort to prevail at an economic and technological level.

This diploma thesis focuses on the development of technology in the means of transport and in particular on autonomous driving. As part of this work, a vehicle was built with the addition of a robotic arm. With the development of a lane recognition algorithm, it became possible to autonomously drive the vehicle into a delimited lane, in real time. The algorithm development was made based on older techniques that were mostly tested on video than in real path movement. The present study is a trigger for future applications and innovations including the use of the robotic arm.

Keywords:

Autonomous Driving, Autonomous Vehicle, Robot, Robotic Arm, Real-Time Lane Detection, Lane Detection Algorithm, OpenCV, Gaussian Blur, Canny Edge Detection, Hough Transform.

Πίνακας περιεχομένων

Κατάλογος σχημάτων

Συντομογραφίες

βλπ	βλέπε
κ.λπ.	και λοιπά
κ.α.	και άλλα
κ.ο.κ	και ούτω καθεξής
Κώδ.	Κώδικας
Σχ.	Σχήμα
TEI	Τεχνολογικό Εκπαιδευτικό Ίδρυμα
5G	5th Generation
ADAS	Advanced Driver Assistance System
BPF	Band Pass Filter
CNC	Computer Numerical Control
CPU	Computer Processing Unit
GMM	Gaussian Mixture Model
GPIO	General Purpose Input Output
GPS	Global Positioning System
HAT	HArdware on Top
HSV	Hue Saturation Value
IoT	Internet of Things
LCAS	Lane Change Assistance System
LDWS	Lane Departure Warning System
LIDAR	LIght Detection And Ranging
LKAS	Lane Keeping Assistance System
OpenCV	Open Computer Vision
PUMA	Programmable Universal Manipulation Arm
PWM	Pulse Width Modulation

RADAR	RAdio Detection And Ranging
RANSAC	Random Sample Consensus
RGB	Red Green Blue
ROI	Region Of Interest
USB	Universal Serial Bus

Κεφάλαιο 1

Εισαγωγή

1.1 Αντικείμενο της διπλωματικής

Η ραγδαία οικονομική και τεχνολογική ανάπτυξη και η ταχεία εξέλιξη θεμελιωδών καινοτομιών καθορίζουν τις μεταφορές της σύγχρονης εποχής. Μια τέτοια επαναστατική καινοτομία, η οποία συναντάται ολοένα και περισσότερο, είναι η αυτόνομη οδήγηση των επιβατικών αλλά και εμπορικών οχημάτων. Οι αλγόριθμοι αναγνώρισης λωρίδας αποτελούν θέμα έρευνας και ανάπτυξης για τις σύγχρονες αυτοκινητιστικές εταιρίες που θέλουν να εισέλθουν στον κόσμο της αυτόματης οδήγησης. Αντικείμενο της παρούσας διπλωματικής εργασίας αποτελεί η αυτόνομη οδήγηση ενός ρομποτικού οχήματος σε μια λωρίδα δρόμου. Η μελέτη της διπλωματικής στηρίζεται, αρχικά, στην κατασκευή ενός ρομποτικού οχήματος με την προσθήκη ενός ρομποτικού βραχίωνα και στη συνέχεια, στην υλοποίηση ενός αλγορίθμου μηχανικής όρασης για την αυτόνομη οδήγησή του σε μια λωρίδα κίνησης. Στα πλαίσια της διπλωματικής, πραγματοποιήθηκε η μελέτη ποικίλων ερευνών και των αλγορίθμων που αυτές προτείνουν. Οι μελέτες αυτές αποτέλεσαν την αφορμή και την θεμελίωση του αλγορίθμου που υλοποιήθηκε. Ο προτεινόμενες αλγόριθμος, σε αντίθεση με την πλειοψηφία των προηγούμενων ερευνών που δοκιμάστηκαν σε βιντεοσκοπημένες διαδρομές, αποτελεί εφαρμογή πραγματικού χρόνου, δηλαδή επιτυγχάνει την αυτόνομη οδήγηση του κατασκευασμένου οχήματος σε μια λωρίδα.

1.2 Οργάνωση του τόμου

Το Κεφάλαιο 2 εκτελεί ιστορική αναδρομή για την εξέλιξη των ρομπότ από τη βιομηχανική επανάσταση.

Το Κεφάλαιο 3 κάνει μια εισαγωγή στην αυτόνομη οδήγηση. Στη συνέχεια επιχειρεί την εισαγωγή στην έννοια κάποιων όρων και την λειτουργία ορισμένων αλγορίθμων στους οποίους αναφέρεται η εργασία.

Το Κεφάλαιο 4 είναι μια βιβλιογραφική επισκόπηση των ερευνών και των αλγορίθμων που έχουν υλοποιηθεί στην αυτόνομη οδήγηση των οχημάτων. Εξετάζονται οι διαφορετικές τεχνικές και τα οφέλη που προσδίδουν αλλά και οι αντίστοιχοι περιορισμοί στη χρήση τους.

Το Κεφάλαιο 5 παρουσιάζει τα υλικά που χρησιμοποιήθηκαν για την κατασκευή του οχήματος με τον ρομποτικό βραχίωνα και την συνεργασία μεταξύ τους για την αρμονική λειτουργία του οχήματος.

Το Κεφάλαιο 6 παρουσιάζει τον αλγόριθμο που αναπτύχθηκε για την αυτόνομη οδήγηση του οχήματος και περιγράφει την υλοποίηση του κώδικα για την επίτευξη αυτού του σκοπού.

Το Κεφάλαιο 7 αποτελεί την σύνοψη της παρούσας εργασίας και αναφέρει ιδέες για μελλοντικές επεκτάσεις την διπλωματικής.

Κεφάλαιο 2

Ρομποτικός Βραχίωνας

2.1 Εξέλιξη Ρομποτικής

Η ρομποτική είναι συνυφασμένη με διάφορους τομείς της επιστήμης, της τεχνολογίας και γενικότερα της καθημερινότητας των ανθρώπων. Η ανάγκη για πρόοδο και εξέλιξη, οδήγησε στην δημιουργία ενός από τα μεγαλύτερα επιτεύγματα της ανθρωπότητας, τα ρομπότ ή αλλιώς αυτοματοποιημένες συσκευές. Με την πάροδο των χρόνων επιτυγχάνονται όλο και περισσότερες καινοτομίες στον τομέα της ρομποτικής, οι οποίες καθιστούν τις συσκευές άμεσα διαθέσιμες στο ευρύ κοινό.

2.2 Η συνεισφορά του da Vinci

Από την αρχαιότητα ακόμα, η ύπαρξη μιας αυτοματοποιημένης συσκευής αποτελούσε το όραμα μεγάλων εκπροσώπων των επιστημών, όπως ο Leonardo da Vinci.

Ο θεμελιώδης λίθος της σύγχρονης βιομηχανίας είναι η εξέλιξη του ρομποτικού βραχίωνα. Την παλαιότερη, ίσως, συνεισφορά έχει ο Πρόδρομος της Μηχανικής, Leonardo da Vinci, ο οποίος είχε ασχοληθεί με ρομποτικούς μηχανισμούς από το 1478. Μετά από εκτενή μελέτη της ανθρώπινης κινησιολογίας, σχεδίασε και κατασκεύασε το πρώτο ανθρωποειδές ρομπότ, έναν μεταλλικό ιππότη (Σχ. 2.1), ο οποίος είχε την δυνατότητα μίμησης των ανθρώπινων κινήσεων του σαγονιού, του λαιμού και των χεριών. Τα άνω άκρα είχαν τέσσερις βαθμούς ελευθερίας, αρθρωτούς ώμους, αγκώνες, καρπούς και παλάμες. Ακόμη, υπήρχε συμφωνία κινήσεων ελεγχόμενη από ένα «ενσωματωμένο» προγραμματίσιμο controller μέσα στο στήθος του ανθρωποειδούς.

Σχήμα 2.1: Μια σύγχρονη αναπαράσταση του μεταλλικού Ιππότη του DaVinci [1]

Το ρομπότ αυτό έχοντας πολλαπλούς βαθμούς ελευθερίας, θα αποτελούσε αργότερα την αφετηρία για το τεχνολογικό ενδιαφέρον του ανθρώπου στην ρομποτική κίνηση [1] [30].

2.3 Θεμελιώδεις Νόμοι Ρομποτικής

Αρχικά, ο όρος «ρομπότ» πρωτοχρησιμοποιήθηκε σε ένα θεατρικό έργο του Joseph Capek, το «Rossum's Universal Robots», το οποίο έκανε πρεμιέρα στην Πράγα το 1921. Η λέξη robot προέρχεται από την τσέχικη λέξη «robota» που σημαίνει «σκληρή δουλειά» [1]. Το έργο αυτό επηρέασε αρνητικά την αντίληψη των ανθρώπων για τα ρομπότ, καθώς απεικόνιζε μια διαφορετική πραγματικότητα. Σε αυτό, το ανθρώπινο δυναμικό αντικαθίσταται από μηχανές σε διάφορες πτυχές της καθημερινότητας. Οι μηχανές, τελικά, ξεκίνησαν επανάσταση εναντίον των αφεντικών τους με σκοπό τον αφανισμό του ανθρώπινου είδους. Η εκτέλεση του έργου έγινε το 1921, όμως χρονολογικά αναφερόταν, στην άγνωστη για την εποχή, δεκαετία του 1960, κατά την οποία, πράγματι, έγιναν οι πρώτες χρήσεις ρομπότ για βιομηχανικούς σκοπούς [1]. Λίγο αργότερα, ο συγγραφέας έργων επιστημονικής φαντασίας, Isaac Asimov, βασιζόμενος στις ιδέες του Capek, διατύπωσε για πρώτη φορά το 1942, με το έργο του,

«Runaround» [31], τους τρεις θεμελιώδεις νόμους της ρομποτικής. Σε μεταγενέστερα μυθιστορήματά του, οι νόμοι αυτοί συμπληρώθηκαν από τον μηδενικό νόμο της ρομποτικής. Αυτοί οι νόμοι, οι οποίοι χρησιμοποιούνται μέχρι και σήμερα, είναι οι εξής:

0. Το ρομπότ, τόσο σε ενεργή όσο και σε αδρανή κατάσταση, δεν θα αποτελέσει κίνδυνο για την ανθρωπότητα

1. Το ρομπότ δε θα κάνει κακό σε άνθρωπο, ούτε με την αδράνειά του θα επιτρέψει να τραυματιστεί ανθρώπινο ον, εφόσον αυτό δεν αντιτίθεται στο μηδενικό νόμο.

2. Το ρομπότ πρέπει να υπακούει στις διαταγές που του δίνουν οι άνθρωποι, εκτός αν αυτές έρχονται σε αντίθεση με τον πρώτο νόμο

3. Το ρομπότ οφείλει να προστατεύει την ύπαρξή του, εφόσον αυτή η ενέργεια δεν συγκρούεται με τον πρώτο και τον δεύτερο νόμο [31]

2.4 Από τα ρομπότ στην Βιομηχανική Επανάσταση

Κατά τη διάρκεια της Βιομηχανικής Επανάστασης, η αυξημένη ζήτηση προϊόντων επέφερε την ανάγκη για αύξηση της παραγωγής. Αυτό ήταν και το κίνητρο για την χρήση αυτόματων μηχανών-ρομπότ που θα είχαν ως αποτέλεσμα τη μείωση του κόστους παραγωγής αλλά και την ταυτόχρονη επιτάχυνσή της, προσφέροντας στο κοινό προϊόντα καλύτερης ποιότητας και απαλλάσσοντας τον ανθρώπινο παράγοντα από επικίνδυνες και ανθυγιεινές δουλειές.

Το πρώτο και απλούστερο βιομηχανικό ρομπότ έγινε πραγματικότητα, χάρη στην εφεύρεση των συσκευών με αριθμητικό σύστημα ελέγχου και των μικροκυκλωμάτων. Τα πρώτα ρομπότ δεν είχαν αισθητήρες για την γνώση της κατάστασης του περιβάλλοντος, άρα η κύρια χρήση τους ήταν να επιτελούν πιο απλές δουλειές, όπως Σήκωσε-και-Τοποθέτησε (Pick-And-Place) αντικείμενα. Καθώς η δύναμη και η αντοχή των ρομπότ ήταν ισχυρότερη από αυτή των ανθρώπων, είχε ως αποτέλεσμα να αντικαταστήσουν τους τελευταίους στις πιο βαριές και επικίνδυνες εργασίες. Χρησιμοποιήθηκαν, αρχικά, ρομπότ με υδραυλικό μηχανισμό, που είχαν την απαραίτητη ισχύ για την εκτέλεση εργασιών με βαρύτερα φορτία.

Με το πέρασμα των χρόνων, η χρήση αισθητήρων εξωτερικού περιβάλλοντος κατέστησε τα ρομπότ ικανά να χρησιμοποιηθούν σε εφαρμογές που απαιτούν πιο περίπλοκες κινήσεις,

όπως η συναρμολόγηση και η συγκόλληση. Ακόμη, στη σύγχρονη εποχή βρίσκουν εφαρμογή σε δουλειές μεγαλύτερης ακρίβειας, όπως εγχειρήσεις, 3D Printing και άλλα.

Το πρώτο ρομπότ που ονομαζόταν Unimate, εφευρέθηκε το 1954 από τον, αναφερόμενο και ως, πατέρα της ρομποτικής, George Charles Devol και χρησιμοποιήθηκε σε βιομηχανικές εφαρμογές. Κατόπιν, η εξέλιξη των βιομηχανικών ρομποτικών χεριών σε συνδυασμό με τον ανταγωνισμό μεταξύ των εταιρειών είχε ως αποτέλεσμα την αύξηση της ζήτησής τους σε παγκόσμιο επίπεδο, πράγμα το οποίο με τη σειρά του αποτέλεσε κίνητρο για περαιτέρω έρευνα και τεχνολογική εξέλιξη.

Στα τέλη της δεκαετίας του 1970 και στις αρχές της δεκαετίας του 1980, η ανάπτυξη της τεχνολογίας των βιομηχανικών ρομπότ αφορούσε, κυρίως, μηχανές που επιτελούν εργασίες συναρμολόγησης. Υπήρχε ανάγκη για ρομπότ με μεγαλύτερη επιτάχυνση, ταχύτητα και επαναληπτικότητα ώστε να μειωθούν οι χρόνοι κύκλων. Οι σημαντικότεροι πελάτες των βιομηχανιών παραγωγής ρομπότ, ήταν και είναι οι αυτοκινητοβιομηχανίες και οι χαλυβουργίες, χώροι εργασίας με ανθυγιεινό περιβάλλον.

Ανάλογα με τη χρήση τους τα ρομπότ στις βιομηχανίες, θα μπορούσαν να καταταχθούν σε τρεις κατηγορίες: διαλογή προϊόντων, συναρμολόγηση και διαδικασίες επεξεργασίας [32].

2.5 Ρομποτική Χειρουργική

Η ιδέα ενός ρομποτικού χεριού που μπορεί να μιμηθεί ανθρώπινες κινήσεις έχοντας όμως μεγαλύτερη ακρίβεια, επηρέασε και τον τομέα της ιατρικής, με αποτέλεσμα την εφεύρεση του Programmable Universal Manipulation Arm (PUMA) (Σχ. 2.2). Η πρώτη εφαρμογή ενός ρομπότ σε εγχείρηση έγινε από το PUMA 560 που χρησιμοποιήθηκε για να κατευθύνει την βελόνα σε μια βιοψία εγκεφάλου κατά τη διάρκεια μιας νευρολογικής επέμβασης. Αυτή η πρωτοποριακή μέθοδος άνοιξε και τον δρόμο για την ρομποτική χειρουργική, προσφέροντας πιο ακριβή και σταθερή κίνηση από αυτήν του ανθρώπινου χεριού [33].

Μια διαφορετική προοπτική της χρήσης των ρομπότ, αποτέλεσε η εφεύρεση του ZEUS (Σχ. 2.3), το 1998, με την ιδέα της τηλεχειρουργικής. Σε αυτό το σύστημα, υπάρχουν δύο μηχανήματα: το κύριο, το οποίο χειρίζεται ο χειρουργός και το δευτερεύον που βρίσκεται σε διαφορετική τοποθεσία και είναι αυτό που εκτελεί στην πράξη την εγχείρηση του ασθενή. Η χρήση αυτή δίνει πολλές δυνατότητες για την κάλυψη κάθε είδους ιατρικής ανάγκης από και σε οποιοδήποτε μέρος στον κόσμο με την ανάπτυξη του δικτύου 5G. Αυτό θα αποτελέσει

μελλοντικά μεγάλο μέρος των ιατρικών επεμβάσεων [1].

Η συνεχώς εξελισσόμενη ρομποτική χειρουργική χρησιμοποιείται σήμερα σε πολλούς διαφορετικούς τομείς, με τα ρομπότ της νέας γενιάς να είναι φθηνότερα και μικρότερου μεγέθους. Αυτό οδηγεί στην όλο και ευρύτερη χρήση ρομπότ σε χειρουργικές επεμβάσεις [1].

Σχήμα 2.2: PUMA robot [2]

Σχήμα 2.3: Zeus robot [1]

2.6 Κατασκευή Ρομποτικού Βραχίωνα

Στα πλαίσια της διπλωματικής κατασκευάστηκε ένα ρομποτικό όχημα με τη προσθήκη ενός ρομποτικού βραχίωνα τοποθετημένου επάνω στη βάση του. Η κατασκευή του οχήματος έγινε σε συνεργασία με τον συμφοιτητή-συνάδελφο Χρήστο Ζέρβα, ο οποίος μελετά τις δυνατότητες του ίδιου οχήματος για την εκπόνηση της διπλωματικής του. Ωστόσο, η παρούσα εργασία επικεντρώνεται στην κίνηση-οδήγηση του οχήματος και όχι στη χρήση και την λειτουργία του βραχίωνα.

Στα πρώτα στάδια εκπόνησης της διπλωματικής εργασίας πραγματοποιήθηκε μελέτη για τις διάφορες χρήσεις του βραχίωνα στον τομέα των μεταφορών. Αναφορικά, ο βραχίωνας θα μπορούσε να χρησιμοποιηθεί σε οδικά έργα και οδικές επισκευές. Ένα πολύ ενδιαφέρον αντικείμενο προς ανάπτυξη αποτελεί ο επαναχρωματισμός των γραμμών κυκλοφορίας, καθώς, συνδυάζει την ακολουθία της κάθε γραμμής και την συντονισμένη κίνηση του βραχίωνα πάνω σε αυτές. Μια ακόμα χρήσιμη λειτουργία του βραχίωνα είναι ο έλεγχος διαβατότητας του δρόμου, η απομάκρυνση πιθανών εμποδίων, καθώς και η αυτόματη διάστρωση ασφάλτου στο δρόμο.

Κατά την μελέτη αυτή, θεωρήθηκε αναγκαία η χρήση ενός αισθητήρα LiDAR πολύ μεγάλου κόστους, ο οποίος θα αύξανε αρκετά τον συνολικό προϋπολογισμό του οχήματος. Για τον λόγο αυτό, δεν ήταν εφικτή η χρήση του και κατά συνέπεια η περαιτέρω ανάλυση του στη παρούσα διπλωματική εργασία.

Κεφάλαιο 3

Θεωρητικό Υπόβαθρο

3.1 Αυτοματοποίηση Μέσων Μεταφοράς

3.1.1 Εισαγωγή

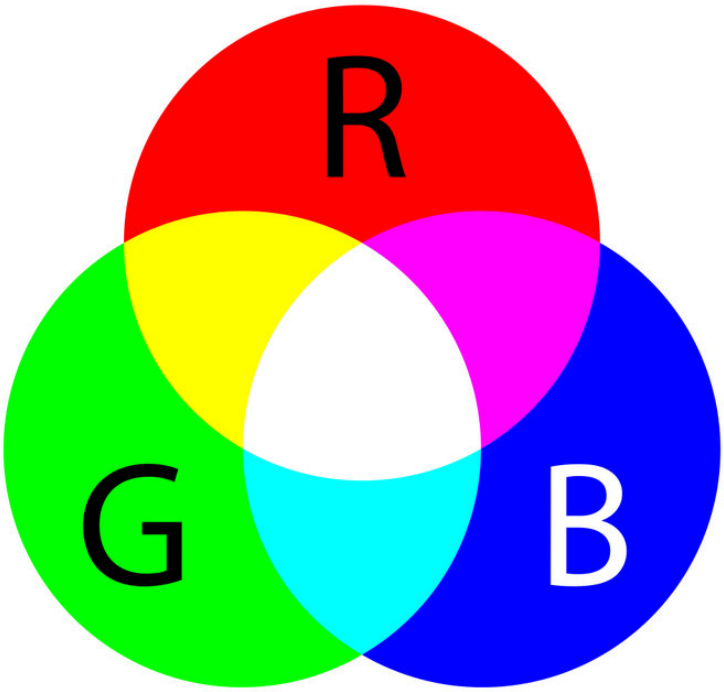
Τα τροχαία ατυχήματα είναι τις περισσότερες φορές αποτέλεσμα ανθρώπινου λάθους όπως η απόσπαση της προσοχής, η απροσεξία και η κακή συμπεριφορά στην οδήγηση [34]. Για αυτό το λόγο, ένα από τα επιτακτικά ζητήματα που απασχολούν πλέον τις αυτοκινητιστικές εταιρείες και τα ερευνητικά ινστιτούτα, είναι να πετύχουν την βελτίωση της ασφάλειας κατά την οδήγηση και την ταυτόχρονη μείωση των τροχαίων ατυχημάτων. Αυτό μπορεί να επιτευχθεί με τη χρήση υπολογιστικών συστημάτων και διαφόρων υποβοηθητικών μεθόδων [21]. Με τη πρόοδο της τεχνολογίας και την δυνατότητα εκτέλεσης πολύπλοκων αλγορίθμων σε πραγματικό χρόνο, έχουν αναπτυχθεί και μελετηθεί διάφοροι τρόποι και τεχνικές για την αναγνώριση της λωρίδας κυκλοφορίας, καθώς και άλλων οχημάτων που κινούνται σε αυτή, έτσι ώστε να πετύχουν έγκαιρη ειδοποίηση του οδηγού για ”αναχώρηση” από την λωρίδα αλλά και άμεση επέμβαση σε περιπτώσεις επερχόμενης σύγκρουσης [17]. Η ουσιαστική γνώση του δρόμου, που επιτυγχάνεται κυρίως με την αναγνώριση και τη συνεχή παρακολούθηση της λωρίδας, είναι ζωτικής σημασίας για την βοήθεια αποφυγής ανθρώπινου σφάλματος. Η αναγνώριση της λωρίδας είναι θεμελιώδης τεχνική για τα Προηγμένα Συστήματα Βοήθειας Οδηγού (Advanced Driver Assistance System [ADAS]) όπως το Σύστημα Προειδοποίησης Αναχώρησης Λωρίδας (Lane Departure Warning System[LDWS]), το Σύστημα Βοήθειας Αλλαγής Λωρίδων (Lane Change Assistance System[LCAS]) και το Σύστημα Βοήθειας Διατήρησης Λωρίδας (Lane Keeping Assistance System[LKAS]) για κεν-

τράρισμα του οχήματος στη λωρίδα [20] [19]. Η βασικότερη σήμανση για την ύπαρξη ενός έξυπνου συστήματος αυτοματοποίησης οποιουδήποτε μέσου μεταφοράς, είναι οι διαχωριστικές γραμμές των λωρίδων του δρόμου. Αυτές θα καθορίζουν και την σωστή πορεία των οχημάτων στον δρόμο αποτρέποντας την επικίνδυνη οδήγηση [18].

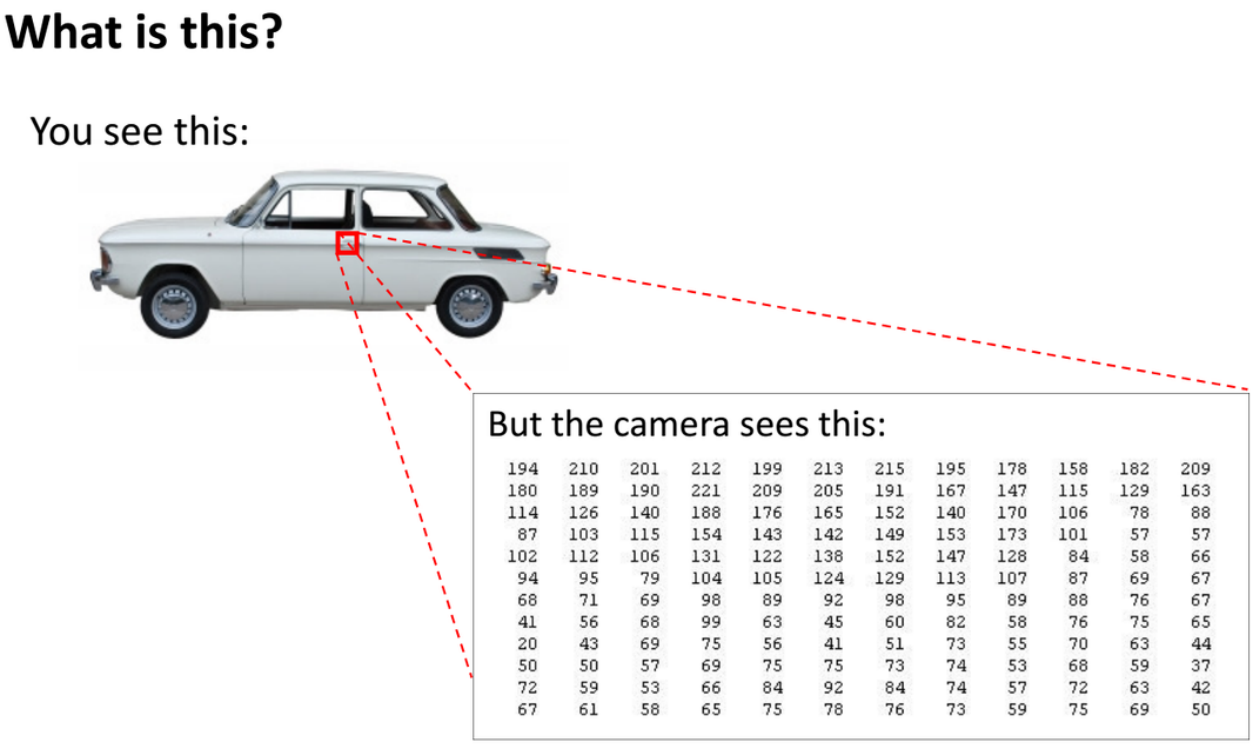
3.1.2 Μηχανική Όραση

Ένας τομέας εξέχουσας σημασίας για την αυτοματοποίηση στη βιομηχανία είναι η Μηχανική Όραση. Αυτή μπορεί να παρέχει ποιοτικό και αυτοματοποιημένο έλεγχο και βασίζεται στην τεχνολογία και την εμπειρία [duth]. Επί της ουσίας, η επιστήμη της μηχανικής όρασης είναι η αναπαράσταση της ανθρώπινης όρασης με βάση κάποιους αλγορίθμους. Ένας ηλεκτρονικός υπολογιστής ή ακόμα και ένα ρομπότ μπορούν πλέον να αποτελέσουν ένα σύστημα μηχανικής όρασης. Η αναπαραγωγή αυτών των αλγορίθμων πραγματοποιείται λαμβάνοντας δεδομένα ψηφιακών εικόνων με την μορφή εικονοστοιχείων (pixel). Η διαδικασία αυτή πραγματοποιείται είτε με τη χρήση απλής και οικονομικής ψηφιακής κάμερας είτε με τη χρήση σύνθετων και πιο ακριβών αισθητήρων, όπως ραδιοεντοπιστές (RAdio Detection And Ranging [RADAR]), στερεοσκοπικές κάμερες και συσκευές εκπομπής παλμικής ακτινοβολίας λέιζερ (LIght Detection And Ranging [LIDAR]).

Πιο συγκεκριμένα, η Μηχανική Όραση επεξεργάζεται εικόνες που λαμβάνει με τη μορφή πληροφορίας RGB. Ως RGB εννοείται ο συνδυασμός τριών αριθμών, οι οποίοι αναπαριστούν την σύνθεση των τριών συνιστωσών των χρωμάτων Κόκκινου, Πράσινου και Μπλε (Red Green Blue [RGB]) (Σχ. 3.1). Μέσω αυτών μπορεί να αναπαραχθεί οποιοδήποτε άλλο χρώμα. Ο τρόπος με τον οποίο η Μηχανική Όραση αντιλαμβάνεται μια εικόνα είναι με τη μορφή πίνακα αριθμητικών τιμών RGB (Σχ. 3.2). Ειδικότερα, η Μηχανική Όραση προσπαθεί να κάνει το αντίστροφο από αυτό που κάνει μια κάρτα γραφικών σε ένα ηλεκτρονικό παιχνίδι. Η τελευταία αναλύει, μετατρέπει και προβάλλει ένα τρισδιάστατο χώρο σε μία επίπεδη εικόνα. Ενώ η Μηχανική Όραση βασιζόμενη στις πληροφορίες που θα πάρει από τις επίπεδες εικόνες, αντιλαμβάνεται το σχήμα και τις ιδιότητες του τρισδιάστατου κόσμου που αυτές απεικονίζουν.



Σχήμα 3.1: Το πρότυπο χρώματος RGB [3]



Σχήμα 3.2: Όραση Υπολογιστή [4]

Μια βασική μεθοδολογία που συναντάται σε μια εφαρμογή μηχανικής όρασης είναι η παρακάτω:

1. Λήψη της πληροφορίας μιας ψηφιακής εικόνας
2. Επεξεργασία της πληροφορίας
3. Ανάλυση της πληροφορίας
4. Εξαγωγή δεδομένων πολλαπλών διαστάσεων [35]

Η Μηχανική Όραση συναντάται σήμερα σε μια ποικιλία εφαρμογών όπως:

- Εφαρμογές αυτοκινήτου: αποφυγή τροχαίων ατυχημάτων και αυτονομία οχημάτων.
- Όραση Ρομπότ: ανίχνευση και αποφυγή εμποδίων, αυτόματος καθορισμός θέσης, αλληλεπιδραση του ρομπότ με τον άνθρωπο.
- Μέτρα Ασφάλειας: δαχτυλικά αποτυπώματα, αναγνώριση χαρακτηριστικών προσώπου και ίριδα.
- Δημιουργία τρισδιάστατου μοντέλου κτιρίων από αεροφωτογραφίες (φωτομετρία).
- Ιατρική απεικόνιση: ανίχνευση καρκινικής μάζας, ρομποτική χειρουργική, αναπαράσταση ανθρώπινων οργάνων.
- Αυτοματισμοί Βιομηχανικού επιπέδου: ανίχνευση βλάβης και επιθεώρηση, αναγνώριση και ταξινόμηση εμπορευμάτων [36].

3.1.3 Τεχνικές Αναγνώρισης Λωρίδας

Οι τεχνικές Αναγνώρισης Λωρίδας (Lane Detection) μπορούν να διακριθούν σε δύο κατηγορίες:

1. Με βάση τα χαρακτηριστικά (feature-based systems)
2. Με βάση τα μοντέλα αναγνώρισης (model-based systems)

Η διαδικασία Αναγνώρισης Λωρίδας είναι ένα πρόβλημα αρκετά απαιτητικό, ειδικότερα όταν ζητείται η επίλυσή του σε πραγματικό χρόνο. Η ασάφεια των προτύπων της λωρίδας, η χαμηλή ορατότητα των γραμμών λόγω ακραίων συνθηκών φωτεινότητας, οι σκιές και οι αντανακλάσεις φωτισμού αλλά και η γενικότερη έλλειψη σήμανσης των λωρίδων κυκλοφορίας,

είναι κάποια από τα καθημερινά σενάρια που δικαιολογούν την πολυπλοκότητα του συστήματος.

Τα συστήματα που βασίζονται σε τεχνικές αναγνώρισης χαρακτηριστικών (feature-based systems) διακρίνουν τις διαχωριστικές γραμμές του οδοστρώματος με κριτήριο την ανίχνευση των ακμών, δηλαδή στοχεύουν στη κατάτμηση της εικόνας απομονώνοντας τμήματα παραβολικών ή ευθειών γραμμών από τις διαχωριστικές γραμμές, την άκρη του δρόμου και τα πεζοδρόμια. Γίνεται μια προεργασία της εικόνας με ενίσχυση της αντίθεσης και χρήση αλγορίθμων για την αναγνώριση ακμών. Στη συνέχεια οι γραμμές της λωρίδας αναγνωρίζονται με χρήση μετασχηματισμού Hough ή άλλους αλγόριθμους. Τα feature-based συστήματα, σε ιδανικές συνθήκες, είναι ικανά να αναγνωρίσουν την λωρίδα κυκλοφορίας με μεγάλη αποτελεσματικότητα. Ωστόσο, σε περιπτώσεις χαμηλής ποιότητας σήμανσης λωρίδας ή οπτικού θορύβου είναι λιγότερο έμπιστα ή και ανίκανα για αναγνώριση.

Αντιθέτως, τα συστήματα που βασίζονται σε τεχνικές αναγνώρισης μοντέλων (model-based systems), μπορούν να αναπαραστήσουν τις λωρίδες με τη μορφή παραμέτρων μιας εξίσωσης. Τα συστήματα αυτά έχουν την ικανότητα να αναγνωρίσουν και να αποτυπώσουν τις γραμμές, ακόμα και σε περιπτώσεις ακραίου φωτισμού. Οι μέθοδοι που ακολουθούν, μοντελοποιούν τη γραμμή με βάση προκαθορισμένα μοντέλα, όπως μοντέλα ευθείας γραμμής, παραβολικής γραμμής και μοντέλα τμηματικών γραμμών (splines) [21] [22].

Τέλος, υπάρχουν ορισμένες μελέτες που για να επιτύχουν αναγνώριση λωρίδας βασίζονται σε τεχνικές μηχανικής μάθησης που αποσκοπούν στην εκπαίδευση ενός νευρωνικού δικτύου. Η εκπαίδευση αυτή εφαρμόζεται σε πολλές παρόμοιες συνθήκες λωρίδων κυκλοφορίας.

3.2 Αλγόριθμοι που χρησιμοποιήθηκαν

3.2.1 Γλώσσα Προγραμματισμού Python

Η Python είναι μια γλώσσα προγραμματισμού που είναι ικανή να διεξάγει οποιοδήποτε είδος εξέτασης πληροφοριών στους τομείς των Επιστημών, των Κοινωνικών Επιστημών, του Εταιρικού κόσμου και της Διοίκησης. Τα μαθηματικά θεωρούνται ως επί το πλείστον η διάλεκτος της επιστήμης, ενώ η ανάλυση πληροφοριών είναι η διάλεκτος της έρευνας. Η έρευνα είναι ζωτικής σημασίας για την ανθρώπινη πρόοδο, οπότε όσο υπάρχουν αναπάντητα ερωτήματα και επιστημονικά αινίγματα τόσο μεγαλώνει η ανάγκη για επίλυση και ανάλυση

περαιτέρω δεδομένων. Αυτή η απλή αλλά ταυτόχρονα ισχυρή γλώσσα προγραμματισμού δημιουργήθηκε από τον Guido van Rossum, το 1989 [37].

Οι δυνατότητες και οι επιλογές που προσφέρει η Python στον προγραμματιστή είναι πολλές, κάποιες από τις οποίες είναι οι εξής:

1. Η δημιουργία ηλεκτρονικών παιχνιδιών.
2. Η οπτικοποίηση δεδομένων και η ανάλυσή τους σε μορφή διαγραμμάτων.
3. Η αναγνώριση φωνής και χαρακτηριστικών προσώπου.
4. Η αυτοματοποίηση απλών και καθημερινών διεργασιών, όπως αυτόματη απάντηση και αποστολή ηλεκτρονικών μηνυμάτων (email).
5. Η ανάπτυξη διαδικτυακών εφαρμογών με χρήση των πλαισίων λογισμικού (frameworks) Djanko και Flask.
6. Η εκπαίδευση μοντέλων νευρωνικών δικτύων [37].

Εδώ και τουλάχιστον 20 χρόνια, η παγκοσμίου χρήσης γλώσσα προγραμματισμού Python, έχει εφαρμοστεί με επιτυχία σε πολλούς τομείς διαφορετικών απαιτήσεων, όπως στον τομέα των υπηρεσιών, στην βιομηχανία και στην επιστημονική έρευνα. Καθώς αποτελεί μια μαθησιακά εύκολη γλώσσα προγραμματισμού τα όρια μεταξύ ενός απλού χρήστη και ενός επαγγελματία προγραμματιστή δεν είναι εύκολα διακριτά. Αυτό καθιστά ότι ένας απλός χρήστης δεν είναι απαραίτητο να ασχολείται επαγγελματικά με τις τεχνητές γλώσσες υπολογιστή. Εξαιτίας της ανάγκης για επίλυση σύνθετων τεχνικών προβλημάτων, ο αριθμός των μηχανικών, των επιστημόνων, των οικονομικών εμπειρογνώμων καθώς και άλλων που χρησιμοποιούν την Python, ακόμα και με ελάχιστη εμπειρία και γνώση προγραμματισμού, συνεχώς και αυξάνεται.

3.2.2 Εισαγωγή στην OpenCV

Η OpenCV είναι μια ελεύθερα προσβάσιμη βιβλιοθήκη ανοιχτού κώδικα (open-source), η οποία χρησιμοποιείται σε Windows, Linux και Mac OS X και είναι γραμμένη σε C και C++. Αναπτύχθηκε, αρχικά, με πρωτοβουλία της Intel Research, στα πλαίσια μιας έρευνας που αφορούσε τις εφαρμογές επεξεργαστών (CPU) υψηλής έντασης. Στη συνέχεια, εξελίχθηκε στο πιο δημοφιλές μέσο για τη διάθεση της τεχνητής όρασης σε παγκόσμιο επίπεδο.

Η βιβλιοθήκη της OpenCV περιλαμβάνει περισσότερες από 500 συναρτήσεις που καλύπτουν σε πραγματικό χρόνο εφαρμογές σε ποικίλους τομείς της μηχανικής όρασης, όπως ιατρική απεικόνιση, στερεοφωνική όραση και ρομποτική, εργοστασιακή επιθεώρηση προϊόντων κ.α. [36].

Με δεδομένο ότι οι συνολικές δυνατότητες της OpenCV είναι διαθέσιμες στο ευρύ κοινό σε συνδυασμό με την απεριόριστη εφευρετικότητα και δημιουργικότητα του ανθρώπινου εγκεφάλου, καθίσταται δυνατή η χρήση της σε αναρίθμητες εφαρμογές, είτε απλές και καθημερινές είτε πιο εξειδικευμένες.

Κάποιες από αυτές τις εφαρμογές είναι η χρήση της σε:

- Ηλεκτρονικά παιχνίδια.
- Τεχνικές συρραφής εικόνων από δορυφόρους και διαδικτυακούς χάρτες.
- Αναλύσεις βιοϊατρικής.
- Αυτόνομη οδήγηση, σε οχήματα ξηράς αλλά και ιπτάμενα και υποβρύχια οχήματα.
- Στρατιωτικές εφαρμογές.
- Συστήματα παρακολούθησης.
- Βιομηχανικό έλεγχο παραγωγής.
- Συστήματα αναγνώρισης προσώπου και αντικειμένων [36].

3.2.3 Τμηματοποίηση Εικόνας

Μια εικόνα αποτελεί έναν συνδυασμό περιοχών και αντικειμένων, από την ανάλυση της οποίας μπορούν να αντληθούν σημαντικές πληροφορίες. Μία από τις πιο απαιτητικές φάσεις κατά την διαδικασία επεξεργασίας μιας εικόνας είναι η τμηματοποίηση της (image segmentation), καθώς το αποτέλεσμα αυτής επηρεάζει άμεσα την συνολική ανάλυση της εικόνας. Κατά την τμηματοποίηση μιας εικόνας πραγματοποιείται ταξινόμηση όλων των εικονοστοιχείων της εικόνας(pixel) σε διαφορετικές ομάδες που παρουσιάζουν παρόμοια χαρακτηριστικά. Κριτήρια αυτής της ταξινόμησης είναι η υφή, το χρώμα, η κίνηση κ.α. [5].

Η τμηματοποίηση μιας εικόνας μπορεί να γίνει είτε σε ασπρόμαυρη μορφή είτε σε έγχρωμη μορφή, αποδίδοντας δύο τελείως διαφορετικά μεταξύ τους αποτελέσματα. Ένα ακόμα

κριτήριο διάκρισης είναι το εύρος κάλυψης που παρέχει, μερικό ή ολικό. Η μερική τμηματοποίηση αφορά ένα μικρό τμήμα της εικόνας, σε αντίθεση με την ολική τμηματοποίηση που αφορά το σύνολο της εικόνας [5].

Για τη διαδικασία τμηματοποίησης υπάρχουν οι εξής εφτά διαφορετικές μεθοδολογίες:

1. Μέθοδος Ακμών (Edge-based)
2. Αναγνώριση Κατωφλίου Ιστογράμματος(Histogram Thresholding)
3. Προσέγγιση Ασάφειας (Fuzzy Approach)
4. Μέθοδος γενετικών αλγορίθμων νευρωνικών δικτύων(Genetic Algorithms)
5. Ομαδοποίηση (Clustering)
6. Μέθοδος Επέκτασης, Διαχωρισμού και Σύμπτυξης Περιοχών
7. Μέθοδος Φυσικού Μοντέλου (Physical Model) [5]

Σε πολλές περιπτώσεις, για την επίτευξη του βέλτιστου προσδιορισμού του περιεχομένου μιας εικόνας, απαιτείται ο διαχωρισμός του αντικειμένου από το φόντο. Η διαδικασία αυτή πραγματοποιείται συνήθως με χρήση δύο τεχνικών, την τεχνική ανίχνευσης ομοιότητας και την τεχνική ανίχνευσης ασυνέχειας. Η πρώτη τεχνική χρησιμοποιείται κυρίως στην τμηματοποίηση έγχρωμων εικόνων και βασίζεται στην μέθοδο Κατωφλίου(Thresholding) σε αντίθεση με την δεύτερη τεχνική που χρησιμοποιείται σε ασπρόμαυρες εικόνες, τις οποίες τμηματοποιεί με κριτήριο τις απότομες εναλλαγές των τιμών των εικονοστοιχείων (pixel) [5].

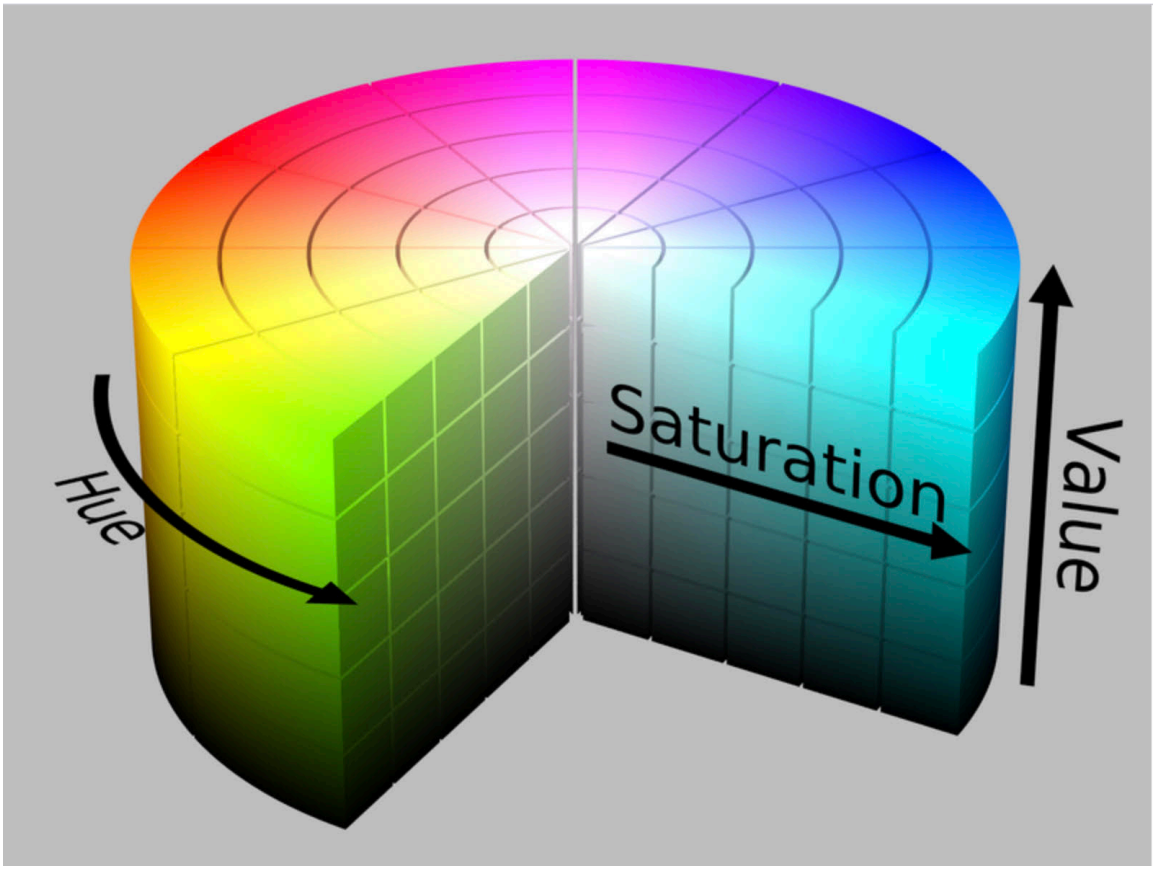
3.2.4 Χρωματικός χώρος HSV

Σύμφωνα με τους Dibya Jyoti Bora και Gupta [5], «χρωματικός χώρος είναι μια συγκεκριμένη οργάνωση χρωμάτων, η οποία σε συνδυασμό με τη χρωματική παλέτα αναπαράστασης των συσκευών, επιτρέπει την αναπαραγωγή των χρωμάτων τόσο σε ψηφιακές όσο και σε αναλογικές αναπαραστάσεις». Η επιλογή του κατάλληλου χρωματικού χώρου ταυτίζεται άμεσα με την διαδικασία της τμηματοποίησης, καθώς αποτελεί το πρώτο ζήτημα που πρέπει να αντιμετωπιστεί. Η χρήση έγχρωμης τμηματοποίησης μπορεί να αποδώσει καλύτερα αποτελέσματα σε σύγκριση με την χρήση ασπρόμαυρης [5].

Ο διαχωρισμός των χρωμάτων πραγματοποιείται με βάση τα εξής χαρακτηριστικά:

- Απόχρωση (Hue)
- Κορεσμός (Saturation)
- Φωτεινότητα (Brightness)
- Διαφάνεια (Lightness)
- Οξύτητα (Chroma)
- Πολυχρωμία (Colorfulness) [5]

Η Απόχρωση, ο Κορεσμός και η Φωτεινότητα (Hue, Saturation, Brightness or Value: [HSV]) αποτελούν τα τρία στοιχεία που διαφοροποιούν το χρώμα από το αχρωματικό φως. Ο ανθρώπινος νους αντιλαμβάνεται και οργανώνει τις διαφορές στα χρώματα με τον παραπάνω τρόπο. Για αυτό το λόγο, στην περιγραφή των χρωμάτων προτιμώνται αυτές οι τρεις έννοιες (HSV) σε σχέση με το μοντέλο αναπαράστασης RGB που χρησιμοποιεί τριάδες αριθμών [38]. Ο χρωματικός χώρος HSV μπορεί να απεικονιστεί ως ένας χρωματικός κύλινδρος



Σχήμα 3.3: Αναπαράσταση HSV [5]

όπως στο Σχ. 3.3, όπου:

- Η η τιμή της Απόχρωσης (Hue) που παίρνει τιμές από 0 έως 2π.
- S η τιμή του Κορεσμού (Saturation) η οποία αντιπροσωπεύει τη ”ζωντάνια” του χρώματος και αυξάνεται όσο απομακρύνεται ακτινικά από το κέντρο με τιμές μεταξύ 0 και 1.
- V η τιμή της Έντασης (Value), η οποία αυξάνει προς τα επάνω παράλληλα με τον κεντρικό άξονα και παίρνει τιμές από 0 έως 100 [5].

Για την μετατροπή του χρωματικού χώρου RGB σε HSV χρησιμοποιούνται οι συναρτήσεις του Σχ. 3.4.

$$
\begin{aligned}
\boldsymbol{V} &= max(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B}) \\
\boldsymbol{S} &= V - min(\boldsymbol{R}, \boldsymbol{G}, \boldsymbol{B})/V \\
\boldsymbol{H} &= \frac{\boldsymbol{G} - \boldsymbol{B}}{6S}, \qquad if \quad \boldsymbol{V} = \boldsymbol{R} \\
\boldsymbol{H} &= \frac{1}{3} + \frac{\boldsymbol{B} - \boldsymbol{R}}{6S}, \qquad if \quad \boldsymbol{V} = \boldsymbol{G} \\
\boldsymbol{H} &= \frac{2}{3} + \frac{\boldsymbol{R} - \boldsymbol{G}}{S}, \qquad if \quad \boldsymbol{V} = \boldsymbol{B}
\end{aligned}
$$

Σχήμα 3.4: Εξισώσεις μετατροπής RGB σε HSV [6]

3.2.5 Ακμές

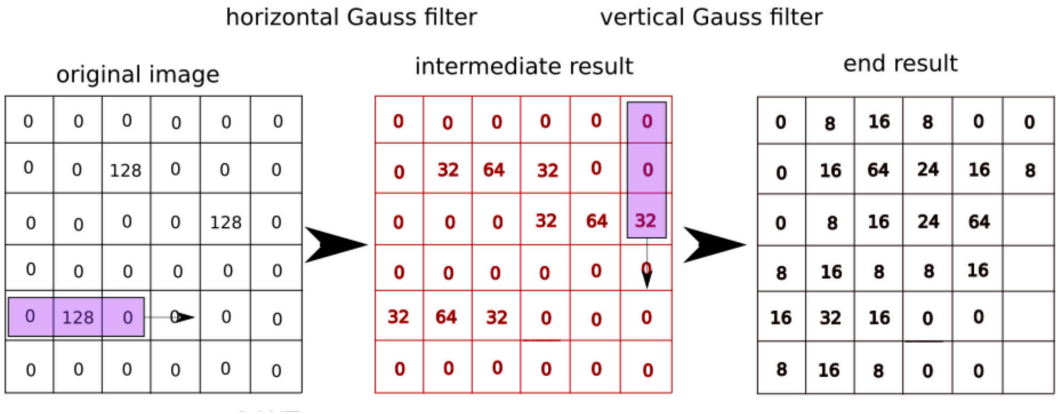
Τον ακρογωνιαίο λίθο στους τομείς της μηχανικής όρασης και στις εφαρμογές ανάλυσης ψηφιακών εικόνων αποτελεί η ανίχνευση ακμών. Ως ακμή ορίζεται η οριοθέτηση των περιοχών που χαρακτηρίζονται από διαφορετικές εντάσεις φωτεινότητας. Πολλές φορές, στην ένταση της φωτεινότητας παρατηρούνται απότομες μεταβολές ή ασυνέχειες οι οποίες οφείλονται σε μεταβολές στο φωτισμό, σε ασυνέχειες στο βάθος και αλλαγές στην υφή και στο είδος του υλικού [39].

Για τον εντοπισμό των σημείων που ανήκουν στις ακμές, χρησιμοποιείται η εύρεση του τοπικού μεγίστου της παραγώγου των τιμών της έντασης, που παρουσιάζεται σε απότομη κλίση ή σε ασυνέχειες στη συνάρτηση της έντασης. Κατά τη διαδικασία αυτή γίνεται εντοπισμός των κύριων εξωτερικών γραμμών σε μια εικόνα, σύμφωνα με το κατώφλι που έχει οριστεί. Κατόπιν, για τον διαχωρισμό των εντοπισμένων εικονοστοιχείων(pixel) της γραμμής

από το φόντο, σε κάθε ένα από αυτά αντιστοιχίζεται ένα δυαδικό ψηφίο. Οι περισσότερες μέθοδοι που εφαρμόζονται για τον εντοπισμό και τη διαχείριση των ακμών χρησιμοποιούν το ίδιο σκεπτικό αποδίδοντας παρόμοια αποτελέσματα [39] [36].

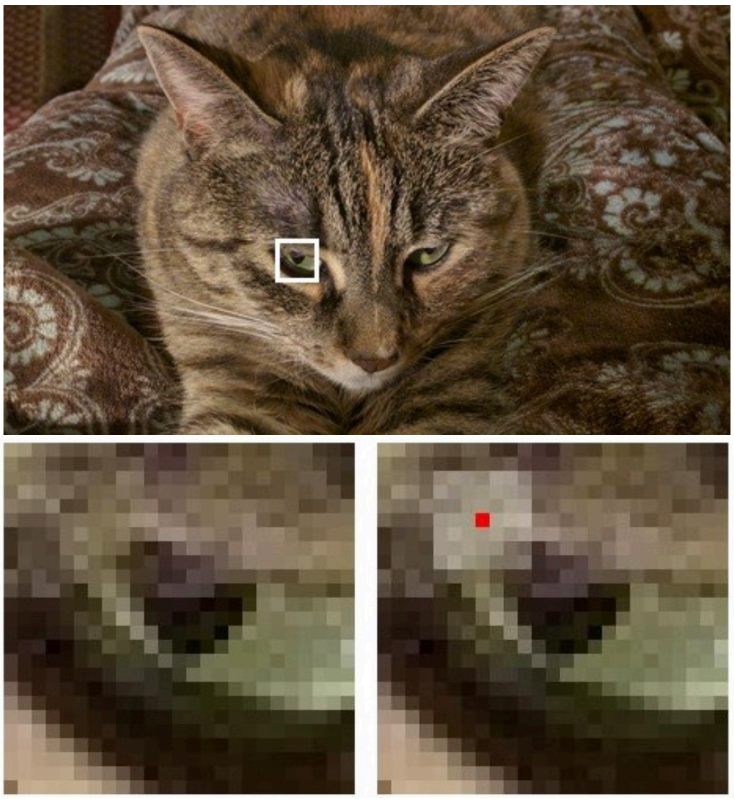
3.2.6 Gaussian Blur

Ένα συχνό φαινόμενο που συναντάται στις φωτογραφίες, και χαρακτηρίζεται ως θόρυβος, είναι τα διάσπαρτα εικονοστοιχεία (pixel) με λανθασμένη χρωματική πληροφορία που περιπλέκουν ή και εμποδίζουν, πολλές φορές, την σωστή και με ακρίβεια ανίχνευση ακμών. Για την ομαλοποίηση της εικόνας, την εξομάλυνση του θορύβου και την αποφυγή ψευδών και παραπλανητικών ακμών, χρησιμοποιούνται φίλτρα θολώματος της εικόνας με πιο διαδεδομένη την τεχνική του Gaussian Blur. Στόχος αυτής της μεθόδου είναι η κλιμακωτή κανονικοποίηση των απότομων αλλαγών των εικονοστοιχείων(pixel). Πιο συγκεκριμένα, εξομαλύνεται μια εικόνα μετριάζοντας τα εικονοστοιχεία τα οποία έχουν ακραίες τιμές σε σχέση με τα γειτονικά τους [8] [40]. Ένα παράδειγμα αποτελεί το Σχ. 3.5, στο οποίο διακρίνονται οι διάσπαρτες ακραίες τιμές 128 σε σχέση με τα γειτονικά τους μηδενικά σημεία.



Σχήμα 3.5: Βήματα Εξομάλυνσης Θορύβου μιας εικόνας με χρήση Gaussian Blur [7]

Κατά την εφαρμογή ενός φίλτρου θολώματος, ο αλγόριθμος επικεντρώνεται σε κάθε εικονοστοιχείο (pixel) της εικόνας ξεχωριστά. Κάθε εικονοστοιχείο που επιλέγεται για τη διαδικασία του θολώματος, περιβάλλεται από μια ομάδα γειτονικών εικονοστοιχείων, που ονομάζεται πυρήνας (kernel), και είναι τετραγωνικός διατηρώντας μονίμως το επιλεγμένο στοιχείο στο κέντρο του [8]. Αυτό μπορεί να γίνει ευκολότερα αντιληπτό στο Σχ. 3.6 όπου το στοιχείο που φιλτράρεται επισημαίνεται με κόκκινο χρώμα.



Σχήμα 3.6: Αναπαράσταση Kernel και του κεντρικού σημείου που φιλτράρεται [8]

Το φίλτρο του Gauss ορίζεται ως ένα ανομοιόμορφο φίλτρο χαμηλής διέλευσης (Low-Pass Filter) που εξομαλύνει τον θόρυβο μειώνοντας τις αμελητέες λεπτομέρειες σε μια εικόνα. Αυτό εφαρμόζεται με τον περιελιγμό (convolution) των εικονοστοιχείων της εικόνας με τον πυρήνα (kernel) που αναφέρθηκε παραπάνω. Η εξίσωση που χρησιμοποιείται για την δημιουργία του δισδιάστατου πυρήνα (kernel) είναι η εξίσωση του Gauss για δύο διαστάσεις που δίνεται από τον τύπο 3.1 [41]:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{3.1}$$

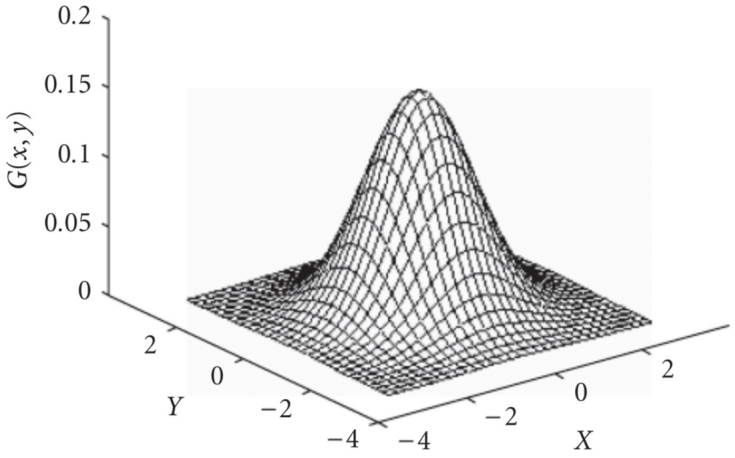
όπου σ είναι η απόκλιση της κατανομής και x,y είναι οι συντεταγμένες του δείκτη.

Πιο αναλυτικά, η νέα τιμή κάθε εικονοστοιχείου (pixel) υπολογίζεται από τον σταθμισμένο μέσο όρο των τιμών χρώματος των γειτονικών εικονοστοιχείων, κατά την εφαρμογή του φίλτρου Gauss. Ο σταθμισμένος μέσος όρος επιλέγεται έναντι του απλού μέσου όρου, καθώς τα εικονοστοιχεία που βρίσκονται πιο κοντά στο κέντρο του πυρήνα (kernel) είναι μεγαλύτερης σημασίας και φέρουν μεγαλύτερο βάρος σε αντίθεση με τα εικονοστοιχεία που βρίσκονται πιο μακριά από το κέντρο. Ένας πυρήνας μεγαλύτερων διαστάσεων (π.χ. 5X5) αποτελείται από περισσότερες τιμές που λαμβάνονται υπόψιν στον μέσο όρο με αποτέλεσμα να θολώνει την εικόνα σε μεγαλύτερο βαθμό από έναν πυρήνα (kernel) μικρότερων διαστά-

σεων (π.χ. 3X3) [8] [40] (Σχ. 3.7 και 3.8).

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Σχήμα 3.7: Η αναπαράσταση του πυρήνα με διακριτές τιμές [9]



Σχήμα 3.8: Η γραφική αναπαράσταση του πυρήνα [9]

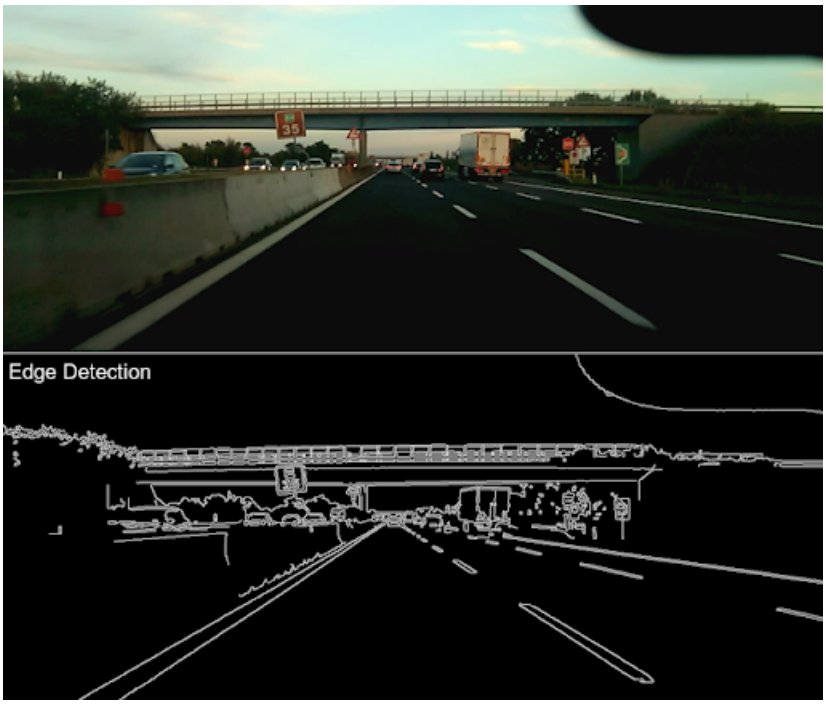
3.2.7 Τελεστής Sobel

Ο τελεστής Sobel δημιουργήθηκε από τον Irwin Sobel, από τον οποίο πήρε και το όνομα του, και αποτελεί έναν διακριτό διαφορικό τελεστή. Με τη χρήση του υπολογίζεται προσεγγιστικά η κλίση της συνάρτησης έντασης μιας εικόνας. Ο παραπάνω τελεστής χρησιμοποιείται σε αλγόριθμους για την ανίχνευση ακμών, δημιουργώντας μια νέα εικόνα στην οποία απεικονίζονται μόνο οι ακμές (Σχ. 3.9). Ο τρόπος λειτουργίας του βασίζεται σε ένα φίλτρο με κάθετη και οριζόντια κατεύθυνση που υπολογίζει την κλίση (πρώτη παράγωγο) της συνάρτησης, η οποία περιγράφει τις τιμές της κάθε σειράς και της κάθε στήλης του πίνακα των εικονοστοιχείων. Ειδικότερα, περιλαμβάνει δύο πυρήνες, τον Gx που εκτιμά την κλίση στην

κατεύθυνση του άξονα x και τον Gy που υπολογίζει την κλίση στην κατεύθυνση του άξονα y [8].

Η τιμή του τελεστή Sobel G δίνεται από τον τύπο 3.2 [5]:

$$G = \sqrt{Gx^2 + Gy^2} \tag{3.2}$$



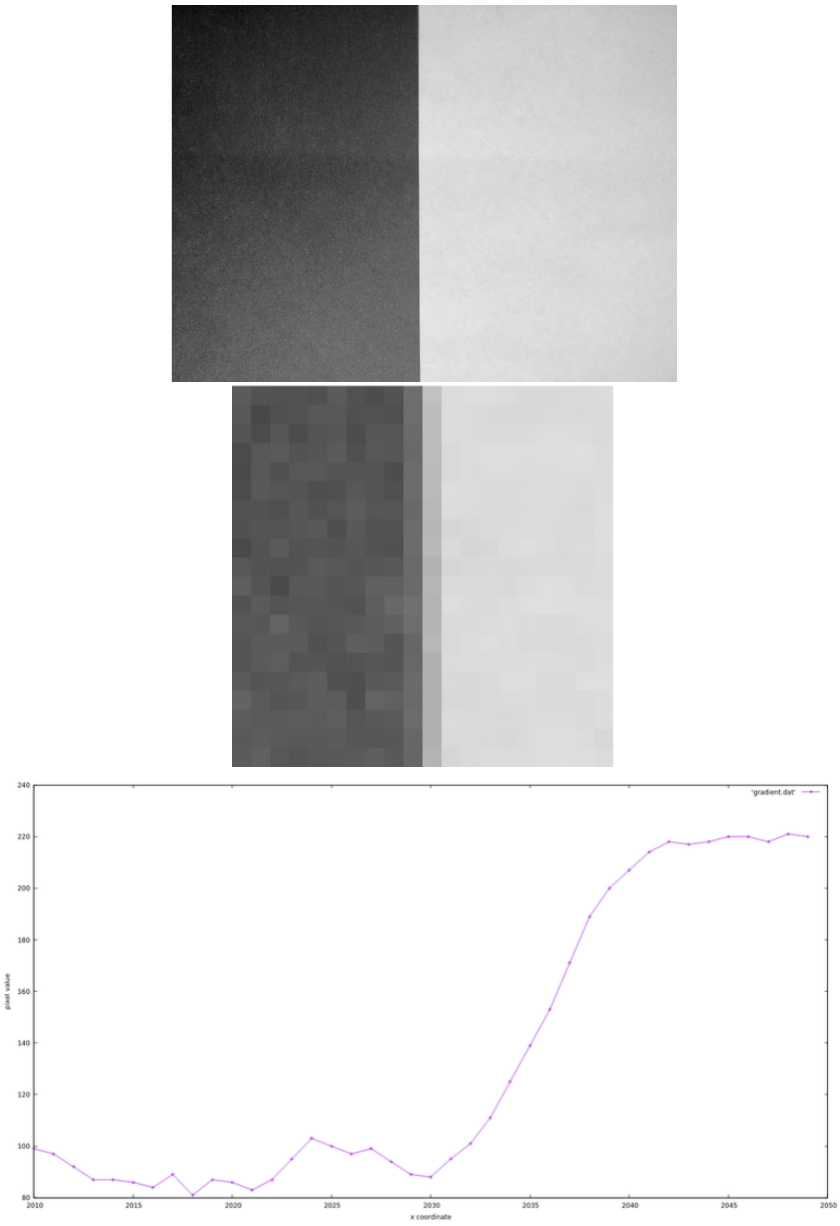
Σχήμα 3.9: Ανίχνευση ακμών σε εικόνα [10]

3.2.8 Ανίχνευση Ακμών με Canny Edge Detection

Για την εξαγωγή της δομής των αντικειμένων μιας εικόνας χρησιμοποιείται η ανίχνευση ακμών, που διευκολύνει στην εστίαση των πιο χρήσιμων χαρακτηριστικών της, όπως το σχήμα, το μέγεθος, ο αριθμός και η σχετική τους θέση. Ο ανιχνευτής ακμών Canny, που δημιουργήθηκε το 1986 από τον John Canny, αποτελεί τον πιο διαδεδομένο αλγόριθμο εύρεσης ακμών [8]. Για τη σωστή λειτουργία του αλγορίθμου κρίνεται απαραίτητο να υπάρχουν σαφή όρια ακμών, να ανιχνεύονται όλες οι πραγματικές τιμές, με εξαίρεση αυτές που δεν υφίστανται και, τέλος, να αποδίδεται μία απόκριση σε κάθε πραγματική τιμή [39].

Όπως παρατηρείται στο Σχ. 3.10 , η προφανής κάθετη ακμή γίνεται αντιληπτή από το ανθρώπινο μάτι, ως μια ξαφνική αλλαγή από σκοτεινά σε φωτεινά εικονοστοιχεία. Στην πραγματικότητα, η αλλαγή αυτή, γίνεται σταδιακά με την ύπαρξη εικονοστοιχείων με ενδιάμεσες

τιμές. Η σταδιακή αυτή μεταβολή, γίνεται εύκολα αντιληπτή από το διάγραμμα στο οποίο διακρίνονται οι τιμές των εικονοστοιχείων μιας οριζόντιας γραμμής.



Σχήμα 3.10: Η μεταβολή της τιμής των εικονοστοιχείων σε μια ακμή [8]

Τα βήματα που ακολουθούνται κατά την εφαρμογή του αλγόριθμου Canny είναι:

1. Απομάκρυνση του θορύβου και εξομάλυνση της εικόνας με χρήση του φίλτρου Gaussian

2. Χρήση του τελεστή Sobel για τον προσδιορισμό της κλίσης της συνάρτησης τιμών των εικονοστοιχείων ως προς τους δύο άξονες, x και y.

Σχήμα 3.11: Πριν και μετά την καταστολή μέγιστων τιμών [11]

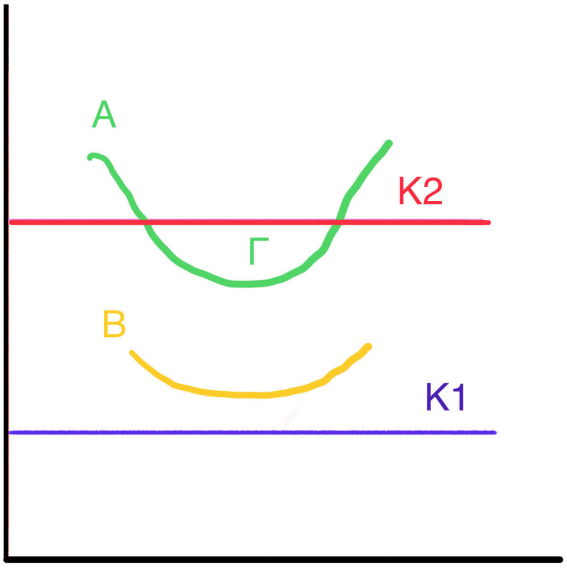
3. Καταστολή των μη μέγιστων τιμών, με αποτέλεσμα την μετατροπή των εντονότερων γραμμών ακμής σε λιγότερο έντονες (Σχ. 3.11). Από κάθε σύνολο γειτονικών εικονοστοιχείων που ορίζουν μια ακμή διατηρείται μόνο αυτό με την μεγαλύτερη τιμή κλίσης, κατά την κατεύθυνση αλλαγής της κλίσης.

4. Ορισμός δύο κατωφλίων (double threshold) K1 και K2, με K1 < K2. Τα εικονοστοιχεία ακμής, των οποίων η κλίση είναι μικρότερη του K1 εξαλείφονται. Τα εικονοστοιχεία ακμής των οποίων η κλίση είναι μεγαλύτερη του K2 θεωρούνται ισχυρά και διατηρούνται. Τέλος, τα εικονοστοιχεία ακμής με τιμή κλίσης ενδιάμεση των δύο κατωφλίων θεωρούνται ασθενείς υποψήφιοι.

5. Τα εικονοστοιχεία που θεωρήθηκαν υποψήφια, θα εξεταστούν περαιτέρω με βάση τα γειτονικά τους, με τη διαδικασία της υστέρησης. Κατά την διαδικασία αυτή επιλέγονται και διατηρούνται μόνο τα εικονοστοιχεία τα οποία συνδέονται με άλλα ισχυρά, ενώ αντίθετα αυτά που δεν συνδέονται εξαλείφονται. Όπως φαίνεται και στο Σχ. 3.12 τα εικονοστοιχεία Α και Γ διατηρούνται, ενώ τα εικονοστοιχεία Β μηδενίζονται [8] [39] [42].

3.2.9 Μετασχηματισμός Hough

Ο μετασχηματισμός Hough, που εφευρέθηκε από τον Paul V. C. Hough, είναι ένας αλγόριθμος που δημιουργήθηκε με σκοπό την εξαγωγή καμπύλων ή ευθείων γραμμών από ένα σύνολο σημείων. Ο μετασχηματισμός Hough, αποτελεί το εργαλείο για την μετατροπή του



Σχήμα 3.12: Διπλή Κατωφλίωση

δισδιάστατου καρτεσιανού χώρου της εικόνας σε έναν νέο δισδιάστατο χώρο, που ονομάζεται χώρος του Hough [43].

Ο χώρος της εικόνας, αποτελείται από ζεύγη πραγματικών αριθμών (x,y) τα οποία αναπαριστούν σημεία. Το σύνολο των σημείων που περιγράφουν μια ευθεία επαληθεύεται από την εξίσωση (3.3):

$$y = ax + b \tag{3.3}$$

Στο χώρο του Hough η θέση κάθε σημείου απεικονίζεται από ένα ζεύγος (a,b), όπου a και b αποτελούν την κλίση και την σταθερά μιας ευθείας στο χώρο της εικόνας, αντίστοιχα [13] [43].

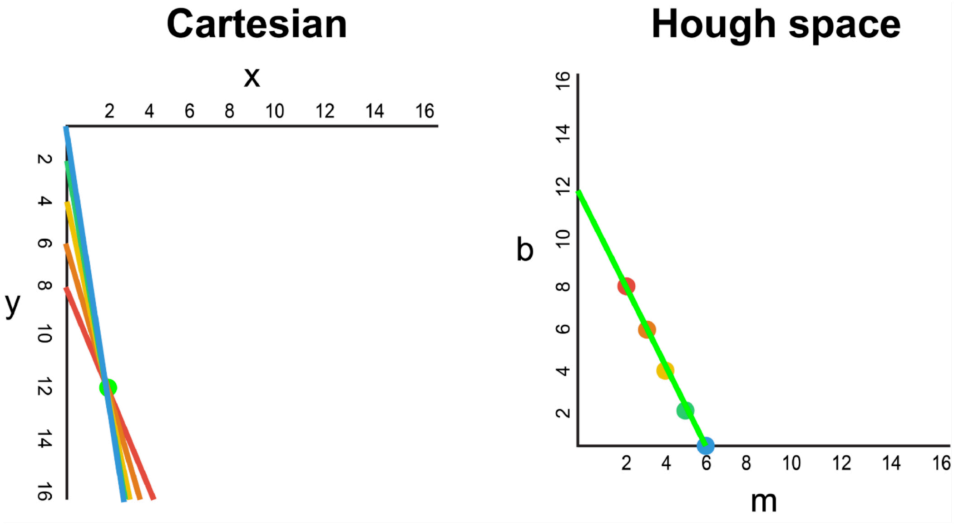
Μέσω του μετασχηματισμού Hough [13]:

1. Κάθε γραμμή στο χώρο της εικόνας αναπαρίσταται με ένα σημείο στο χώρο του Hough, και αντίστροφα (Σχ. 3.13).

2. Κάθε σημείο στο χώρο της εικόνας αναπαρίσταται με μια γραμμή στο χώρο του Hough, και αντίστροφα (Σχ. 3.14).

Ένα πρόβλημα που προκύπτει είναι η αδυναμία αναπαράστασης μιας κάθετης γραμμής καθώς η κλίση της απειρίζεται. Ο τρόπος για την αποφυγή αυτού του προβλήματος είναι η αναπαράσταση της γραμμής στο χώρο της εικόνας με χρήση των πολικών συντεταγμένων, η οποία στο χώρο του Hough περιγράφεται με (r,θ) αντί για (a,b). Στη περίπτωση αυτή, η

Σχήμα 3.13: Γραμμή σε σημείο [12]

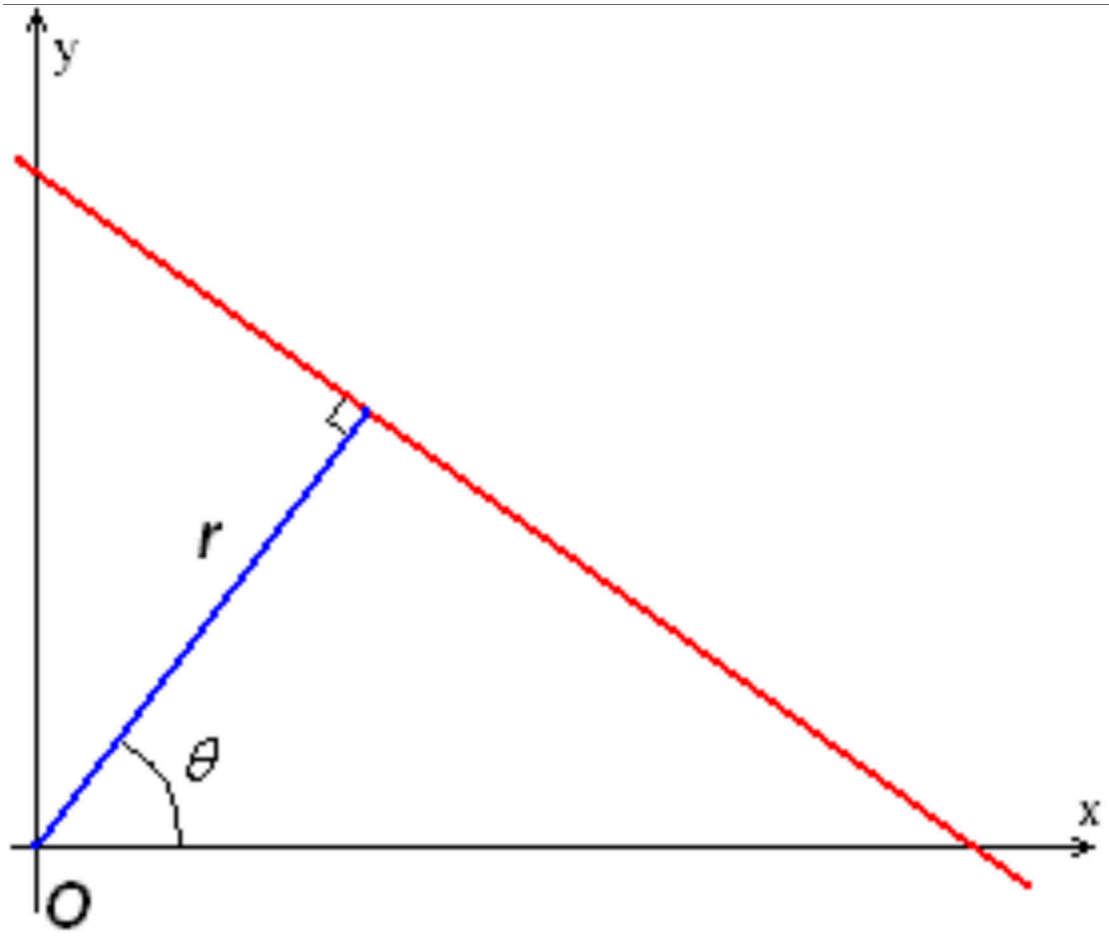


Σχήμα 3.14: Σημείο σε γραμμή [12]

εξίσωση της ευθείας που χρησιμοποιείται είναι η (3.4):

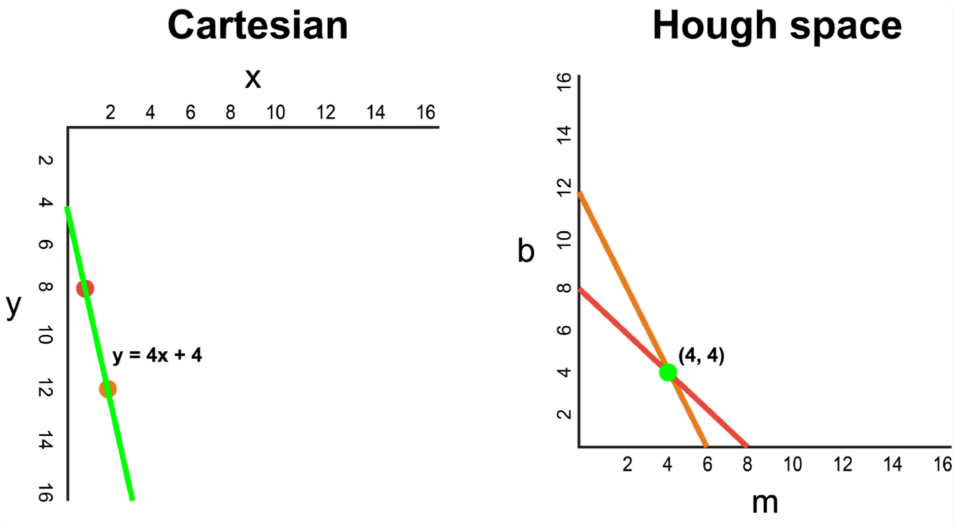
$$r = x\cos\theta + y\sin\theta \tag{3.4}$$

όπου r είναι η κάθετη απόσταση από την αρχή των αξόνων και θ η γωνία αυτής της απόστασης (Σχ. 3.15) [13] [43].

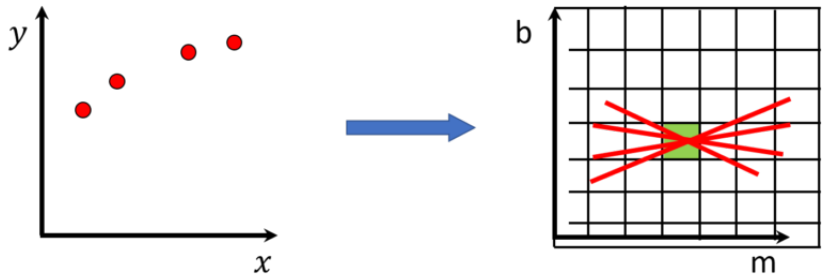


Σχήμα 3.15: Μετατροπή σε πολικές συντεταγμένες [13]

Η διαδικασία του μετασχηματισμού αποτελεί μια εύκολη διαδικασία όταν υπάρχουν δύο ή περισσότερα συνευθειακά σημεία, καθώς το σημείο τομής των δύο ευθειών στο χώρο του Hough αντιπροσωπεύει την ευθεία που περνά από τα δύο σημεία και είναι μοναδικό (Σχ. 3.20). Στην περίπτωση, όμως, που τα σημεία δεν βρίσκονται στην ίδια ευθεία και υπάρχει μια μικρή απόκλιση, ακολουθείται διαφορετική μέθοδος. Ο χώρος του Hough τμηματοποιείται σε τετράγωνα κελιά και έπεται μια διαδικασία ψηφίσματος. Σε αυτήν, εντοπίζονται τα σημεία τομής όλων των ευθειών στο χώρο του Hough και επιλέγεται το κελί που περιέχει τα περισσότερα σημεία τομής. Το κελί αυτό, αποτελεί το ζεύγος τιμών Hough που περιγράφει την μοναδική ευθεία στο χώρο εικόνας (Σχ. 3.17) [13].



Σχήμα 3.16: Εύρεση ευθείας από το σημείο τομής [12]



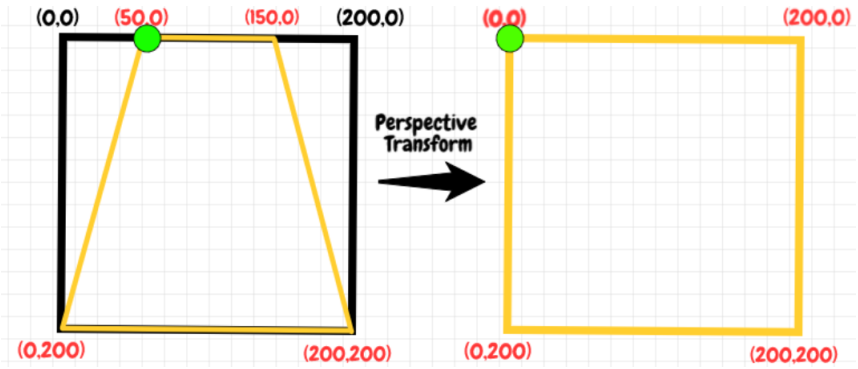
Σχήμα 3.17: Διαδικασία ψήφισης κοινού σημείου τομής [14]

3.2.10 Μετασχηματισμός Προοπτικής - Perspective Transform

Ο μετασχηματισμός προοπτικής συνδυάζει δύο έννοιες σε μία. Ως προοπτική ορίζεται η δυνατότητα του ανθρώπινου οφθαλμού να αντιλαμβάνεται τα αντικείμενα που βρίσκονται σε κοντινή απόσταση ως μεγαλύτερα, ενώ τα αντικείμενα που βρίσκονται σε μακρινή απόσταση ως μικρότερα και συγκλίνοντα. Με την έννοια μετασχηματισμό εννοείται η μετατροπή της κατάστασης ενός αντικειμένου σε άλλη. Με τον μετασχηματισμό προοπτικής μετατρέπεται ένας τρισδιάστατος κόσμος σε μια δισδιάστατη εικόνα, με τον πολλαπλασιασμό των επιμέρους εικονοστοιχείων με ένα πίνακα μετασχηματισμού [44].

Η χρήση του μετασχηματισμού προοπτικής στη μηχανική όραση είναι μια τεχνική που συναντάται συχνά σε διάφορες εφαρμογές, όπως για παράδειγμα οι αλγόριθμοι εύρεσης λωρίδων. Σε αυτήν την περίπτωση, μετατρέπεται η εικόνα από την προοπτική του οχήματος

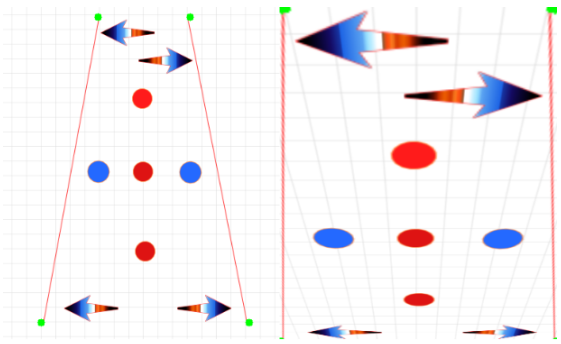
Σχήμα 3.18: Μετασχηματισμός Προοπτικής σε λωρίδα κυκλοφορίας [15]



Σχήμα 3.19: Μετασχηματισμός Συντεταγμένων [16]

στην προοπτική ενός ιπτάμενου πουλιού. Η λωρίδα απεικονίζεται σε κάτοψη, δίνοντας με αυτό το τρόπο καλύτερη οπτική της συνολικής διαδρομής.

Για την μετατροπή αυτήν είναι απαραίτητος ο προσδιορισμός τεσσάρων συντεταγμένων πάνω στην εικόνα από την προοπτική του οχήματος. Οι γνωστές αυτές συντεταγμένες θα πρέπει να μετατραπούν στις τέσσερις νέες και εξίσου γνωστές γωνίες της εικόνας. Αυτό πραγματοποιείται με τον πολλαπλασιασμό των αρχικών συντεταγμένων με έναν πίνακα μετασχηματισμού, ο οποίος προκύπτει από τα παραπάνω αρχικά δεδομένα, και στη συνέχεια χρησιμοποιείται και σε όλα τα περικλειόμενα εικονοστοιχεία [15].



Σχήμα 3.20: Μετασχηματισμός Προοπτικής [16]

Σχήμα 3.21: Μετασχηματισμός Προοπτικής [15]

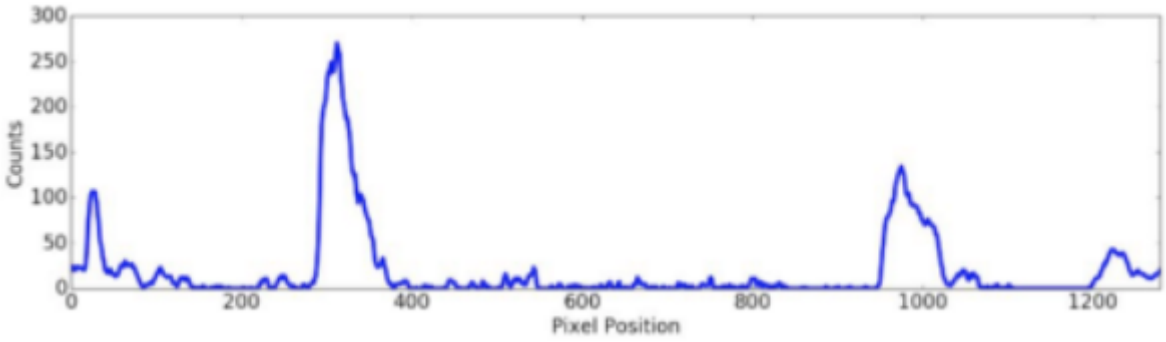
3.2.11 Αλγόριθμος Αναζήτησης Συρόμενου Παραθύρου (Sliding Window Search Algorithm)

Στην αναγνώριση Λωρίδων Κυκλοφορίας, με τον όρο συρόμενο παράθυρο, εννοείται μια ορθογώνια περιοχή σταθερών διαστάσεων, ύψους και πλάτους, που πραγματοποιεί οριζόντιες ολισθήσεις πάνω σε μια εικόνα. Η εφαρμογή της τεχνικής του συρόμενου παραθύρου έχει σκοπό τον εντοπισμό και τον προσδιορισμό της κατεύθυνσης των γραμμών της λωρίδας κίνησης του οχήματος, καθώς επίσης και την διαφοροποίηση της δεξιάς από την αριστερή γραμμή της λωρίδας. Με τον αλγόριθμο αυτόν επιτυγχάνεται η αναγνώριση τόσο ευθειών όσο και καμπύλων γραμμών [19].

Η τεχνική του συρόμενου παραθύρου εφαρμόζεται πάνω σε μία εικόνα στην οποία έχει πραγματοποιηθεί κατωφλίωση των τιμών των εικονοστοιχείων (Thresholding). Κατά την κατωφλίωση διατηρούνται μόνο τα εικονοστοιχεία με τιμές που κυμαίνονται μεταξύ ενός προκαθορισμένου πεδίου ορισμού. Η χρήση αυτής της μεθόδου προτιμάται, καθώς δεν είναι χρήσιμες οι πληροφορίες όλων των εικονοστοιχείων μιας εικόνας δρόμου, αλλά μόνο αυτών που προσδιορίζουν τις γραμμές της λωρίδας (Σχ. 3.21) [15].

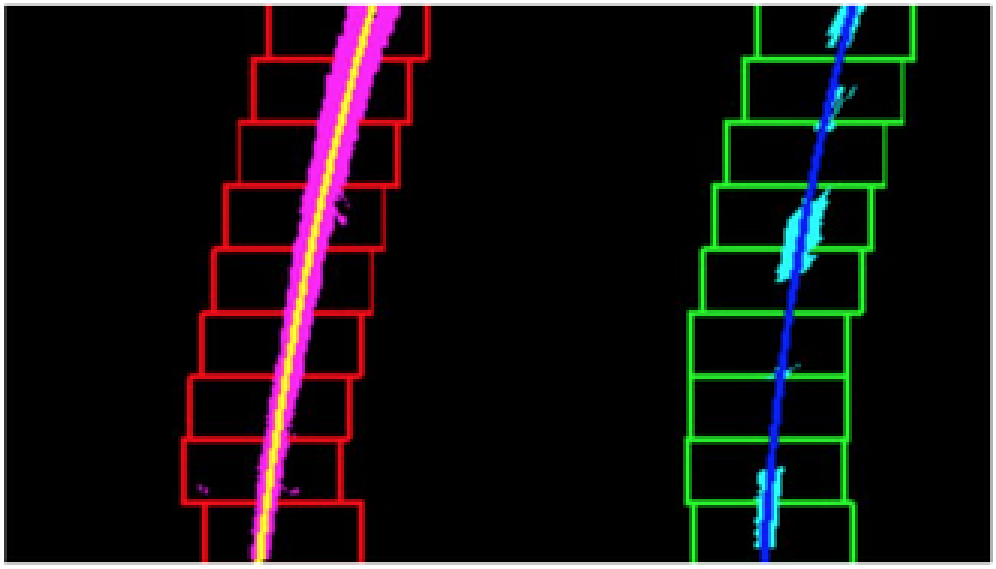
Για την αναζήτηση της αρχικής τοποθεσίας του παραθύρου, συνήθως, χρησιμοποιείται η τεχνική του ιστογράμματος των στηλών. Ως ιστόγραμμα ορίζεται η γραφική παράσταση που υπολογίζει και καταγράφει τη συχνότητα των εικονοστοιχείων που αντιπροσωπεύουν την λωρίδας (Σχ. 3.22).

Μετά τον εντοπισμό των δύο υψηλότερων κορυφών στο ιστόγραμμα, οι οποίες αντι-



Σχήμα 3.22: Υπολογισμός Ιστογράμματος [15]

προσωπεύουν την θέση στον άξονα x των δύο γραμμών της λωρίδας, τοποθετούνται στο χαμηλότερο μέρος της εικόνας δύο παράθυρα στην οριζόντια θέση που υπολογίστηκε. Τα εικονοστοιχεία που βρίσκονται μέσα στο δεξί ή στο αριστερό παράθυρο σημαδεύονται και συγκαταλέγονται στην λίστα των σημείων που αντιπροσωπεύουν την δεξιά και αριστερή γραμμή αντίστοιχα. Στη συνέχεια, με τον υπολογισμό της μέσης συνιστώσας τιμής αυτών των εικονοστοιχείων, ως προς τον άξονα χ, καθορίζεται η θέση του αμέσως επόμενου παραθύρου, στοιβαγμένο στην κορυφή του τρέχοντος. Στη περίπτωση μη αναγνώρισης αρκετών εικονοστοιχείων, το επόμενο παράθυρο έχει την ίδια οριζόντια θέση με το τρέχον. Αυτό επαναλαμβάνεται μέχρι το ανώτερο άκρο της εικόνας. Τέλος, με βάση τις δύο λίστες των εικονοστοιχείων που αντιπροσωπεύουν τις δύο γραμμές της λωρίδας, εφαρμόζεται κάποια μέθοδος για την βέλτιστη προσέγγιση της καμπύλης που σχηματίζουν (Σχ. 3.23) [15] [19]. Παράδειγμα αποτελεί η μέθοδος των ελαχίστων τετραγώνων, η οποία προσεγγίζει βέλτιστα την καμπύλη που διαγράφουν κάποια σημεία στον χώρο, υπολογίζοντας την πολυωνυμική συνάρτηση που την περιγράφει με το μικρότερο δυνατό σφάλμα.



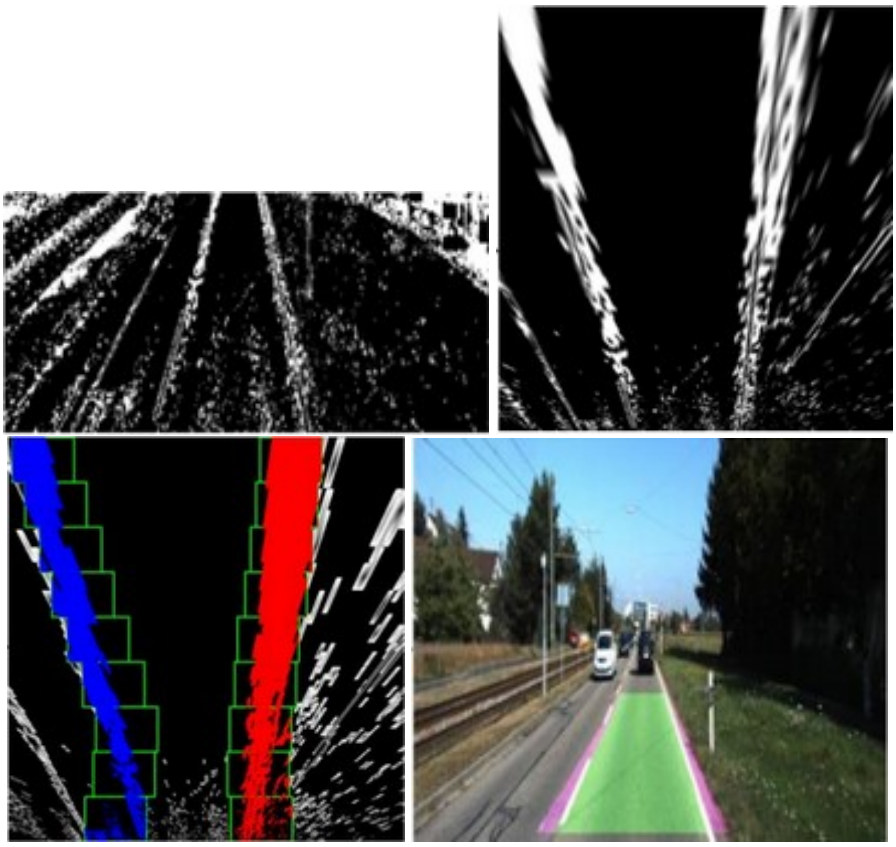
Σχήμα 3.23: Εφαρμογή Συρόμενων Παραθύρων [15]

Κεφάλαιο 4

Βιβλιογραφική Επισκόπηση Προηγούμενων Επιστημονικών Ερευνών

Μία από τις πιο συνηθισμένες προσεγγίσεις για την λύση του προβλήματος αναγνώρισης λωρίδας σε πραγματικό χρόνο είναι η χρήση του μετασχηματισμού προοπτικής και η αναζήτηση συρόμενου παραθύρου. Ο Haque κ.α. [17], έπειτα από χρωματική Κατωφλίωση (HSV Thresholding) και κατωφλίωση με χρήση φίλτρου Sobel για ανίχνευση απότομων αλλαγών στις τιμές των pixel(Gradient Thresholding), εφαρμόζουν μετασχηματισμό της προοπτικής. Στη συνέχεια, με χρήση αναζήτησης συρόμενου παραθύρου (Υποενότητα 3.2.11) τα δύο παράθυρα ξεκινώντας από το κάτω μέρος της εικόνας προς τα πάνω, ανιχνεύουν κομμάτια των γραμμών της λωρίδας (Σχ. 4.1). Η δεξιά λωρίδα ξεχωρίζει από την αριστερή με βάση τις συντεταγμένες των pixel της κάθε γραμμής και υπολογίζεται η καμπύλη δευτέρου βαθμού που εφαρμόζει στην κάθε μία από αυτές. Έπειτα, χρωματίζεται ο χώρος που βρίσκεται ανάμεσα στις γραμμές και γίνεται αντίστροφος μετασχηματισμός προοπτικής (Σχ. 4.1). Ο αλγόριθμός τους, ο οποίος αναπτύχθηκε σε γλώσσα python με χρήση της βιβλιοθήκης OpenCV, δοκιμάστηκε με αποτελεσματικότητα 84% πάνω σε μεμονωμένες εικόνες λωρίδας, σε διάφορα σενάρια φωτισμού και καιρού.

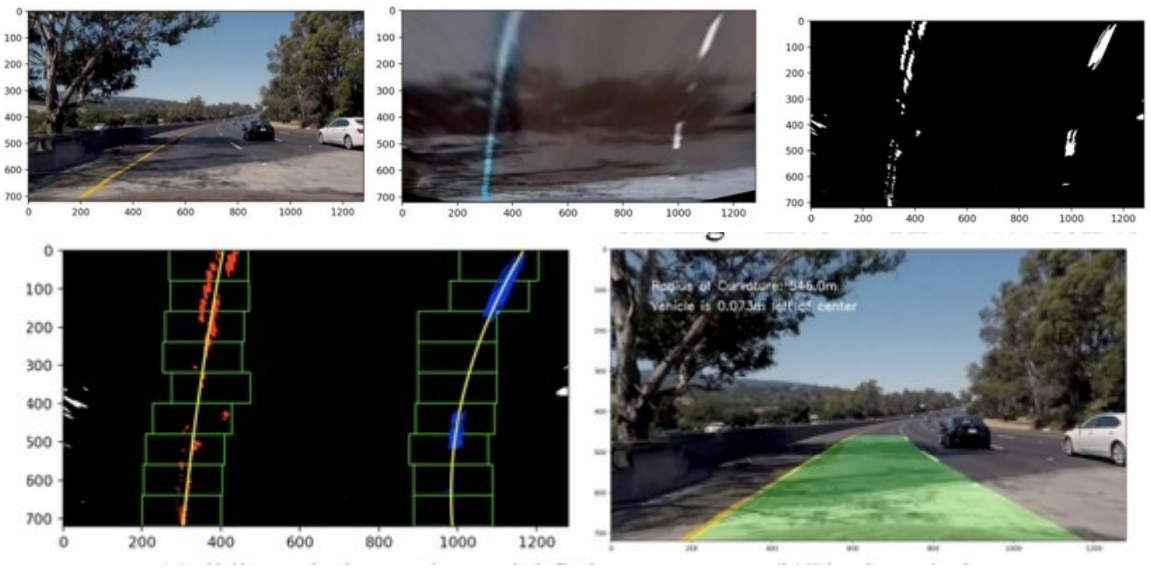
Μία παρόμοια προσέγγιση είχε και ο Ziqiang Sun [18], με την διαφορά ότι έκανε επιπρόσθετη χρήση χρωματικής κατωφλίωσης για αναγνώριση των λευκών και κίτρινων γραμμών ξεχωριστά και εφαρμογή φίλτρου θολώματος Gauss για εξάλειψη του θορύβου. Στη συνέχεια, επιτέλεσε αναγνώριση ακμών με τον αλγόριθμο Canny και εφάρμοσε ένα πλαίσιο ενδιαφέροντος στην εικόνα, ώστε να γίνεται επεξεργασία μόνο των τμημάτων της εικόνας που μπορεί να περιέχουν κομμάτι της λωρίδας, π.χ., το κάτω μέρος. Η συνέχεια είναι ίδια με την



Σχήμα 4.1: Μετασχηματισμός προοπτικής και εφαρμογή Αλγορίθμου Συρόμενων Παραθύρων και τελικό αποτέλσμα [17]

προσέγγιση του Haque, καθώς εφαρμόζεται μετασχηματισμός προοπτικής και αναζήτηση συρόμενων παραθύρων για εύρεση των καμπυλών γραμμών της λωρίδας (Σχ. 4.2). Η αναζήτηση των συρόμενων παραθύρων ξεκινά με τον υπολογισμό του ιστογράμματος στις στήλες της εικόνας για να βρεθούν οι στήλες με τα περισσότερα pixel που ανήκουν σε γραμμή. Τέλος, υπολογίζεται ο βαθμός απόκλισης της πορείας του οχήματος από το κέντρο της λωρίδας και ο χρωματισμός της λωρίδας πάνω στην εικόνα.

Την ίδια τεχνική, αλλά με μικρές διαφορές ακολουθούν οι Dharoori και Narayanan [19], οι οποίοι χωρίς να εφαρμόσουν χρωματική κατωφλίωση επιτέλεσαν διόρθωση της στρέβλωσης της εικόνας λόγω του ευρυγώνιου φακού. Ο φακός αυτός τείνει να στρεβλώνει τις περιοχές κοντά στις γωνίες ώστε να χωρέσουν στον τετράγωνο χώρο της εικόνας. Ως συνέχεια της έρευνας τους, εφαρμόζουν πρώτα τον αλγόριθμο εύρεσης ακμών Canny και έπειτα αναζήτηση συρόμενων παραθύρων με τη βοήθεια ιστογράμματος για να προσεγγίσουν τις καμπύλες γραμμές της λωρίδας. Η προσέγγιση των πολυωνύμων που εκφράζουν τις καμπύλες των γραμμών γίνεται με τη μέθοδο ελαχίστων τετραγώνων. Επιπροσθέτως, γίνεται αντίστρο-

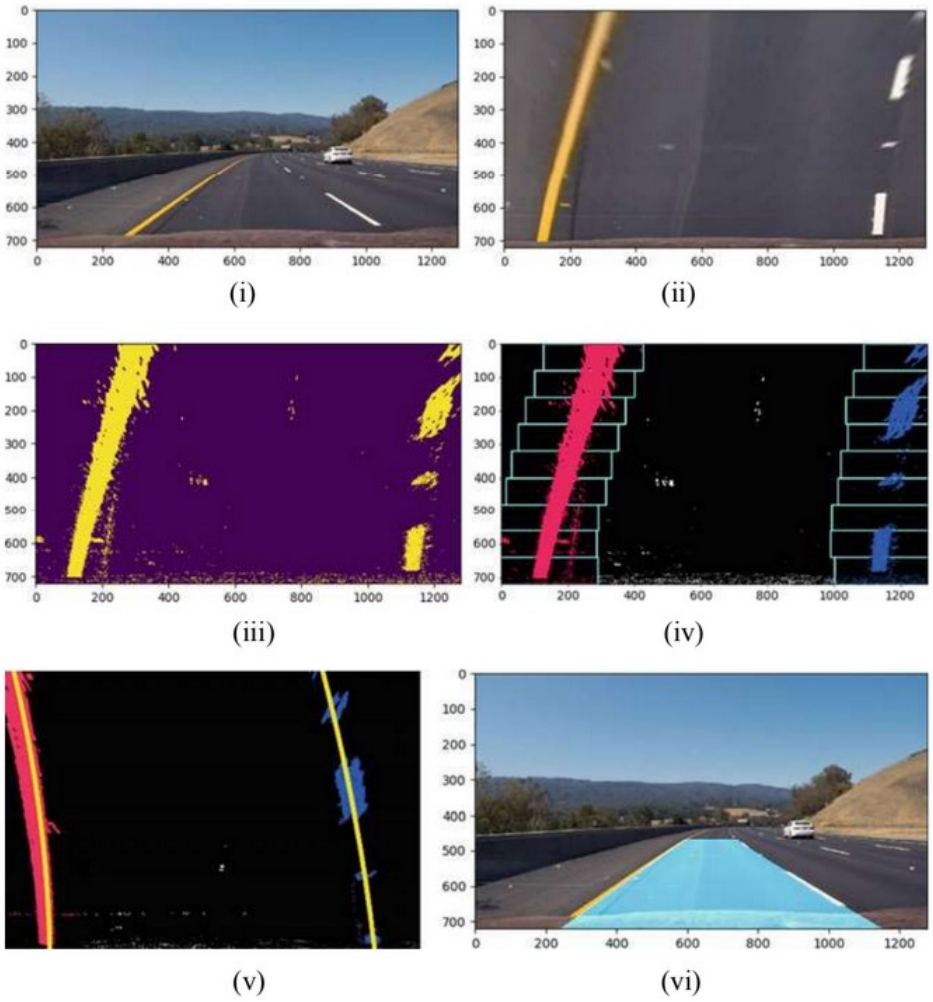


Σχήμα 4.2: Μετασχηματισμός προοπτικής, εφαρμογή Αλγορίθμου Συρόμενων Παραθύρων και τελικό αποτέλσμα [18]

.

φος μετασχηματισμός προοπτικής και χρωματίζεται ο χώρος της αναγνωρισμένης λωρίδας (Σχ. 4.3). Οι τελικές δοκιμές, σε βιντεοσκοπημένες διαδρομές, δείχνουν την ανθεκτικότητα του συστήματος σε διάφορες εναλλαγές του περιβάλλοντος. Το σύστημα, ωστόσο, παραμένει ευάλωτο σε σκιασμούς και απότομες αλλαγές στην υφή του δρόμου.

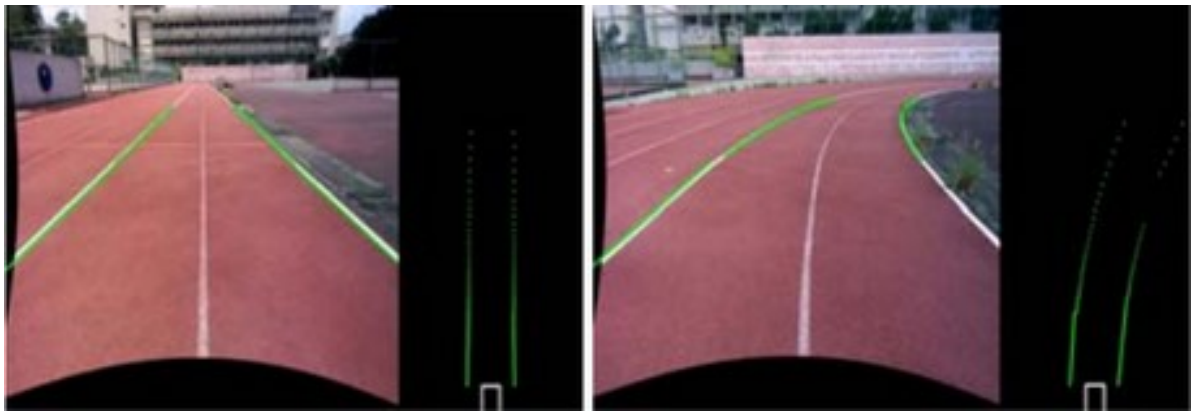
Μια διαφορετική προσέγγιση από τις προηγούμενες, ακολούθησαν οι Huang και Tsai [20], οι οποίοι έφτιαξαν ένα όχημα μεγάλου μεγέθους το οποίο δοκιμάστηκε σε πανεπιστημιακές εγκαταστάσεις. Ο αλγόριθμος που χρησιμοποίησαν περιλαμβάνει: την εφαρμογή πλαισίου ενδιαφέροντος για την διατήρηση ενός τμήματος της εικόνας, μετατροπή της εικόνας σε ασπρόμαυρη, εξάλειψη θορύβου με φίλτρο Gauss και, έπειτα, εύρεση ακμών με μέθοδο Canny. Η αναγνώριση των γραμμών γίνεται με το μετασχηματισμό Hough για την εύρεση των τμημάτων των γραμμών που αναγνωρίζονται. Τέλος, υπολογίζεται η γωνία κλίσης του οχήματος με βάση τη θέση των δύο γραμμών της λωρίδας και πραγματοποιείται αυτόνομη οδήγηση του οχήματος στηριζόμενη στους υπολογισμούς (Σχ. 4.4). Συμπληρωματικά της αυτόνομης οδήγησης, έγινε και χρήση ενός μετρητή αποστάσεως με ακτίνα λέιζερ. Η χρήση του συνετέλεσε στην αποφυγή εμποδίων με απλή παράκαμψη της πορείας ακολουθώντας παράλληλα την λωρίδα. Η έρευνα αυτή Θεωρείται πολύ σημαντική, καθώς δοκιμάστηκε σε αυτοδημιούργητο όχημα σε αντίθεση με τις υπόλοιπες που αναφέρονται σε αυτό το κεφάλαιο που χρησιμοποιούν βιντεοσκοπημένες διαδρομές και φωτογραφίες. Αποτελεί εφαρμογή



Σχήμα 4.3: Στιγμιότυπα Μετασχηματισμού Προοπτικής και εφαρμογής Αλγορίθμου Συρόμενων Παραθύρων και τελικό αποτέλσμα [19]

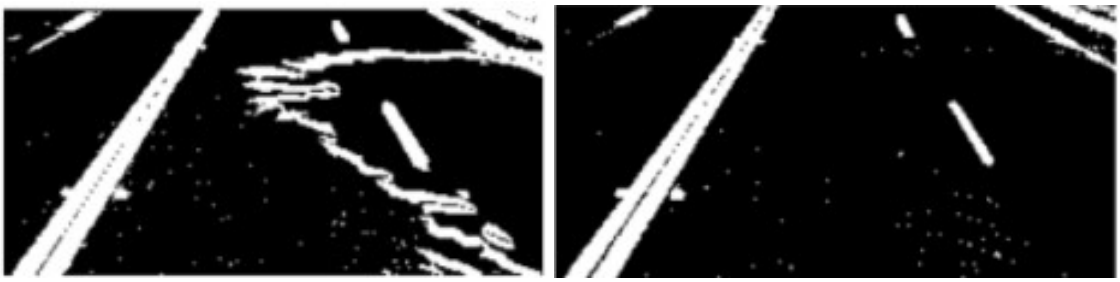
πραγματικού χρόνου και έμπνευση για την προσέγγιση της παρούσας εργασίας.

Ο Xing κ.α. [21] ενισχύει τον παραπάνω αλγόριθμο με έναν συμπληρωματικό που ενεργοποιείται σε περίπτωση ανίχνευσης με χαμηλή απόδοση, δηλαδή σενάρια χαμηλού φωτισμού ή σκίασης του δρόμου. Ο συμπληρωματικός αλγόριθμος περιλαμβάνει δύο αλγόριθμους μηχανικής μάθησης. Ο πρώτος ονομάζεται GMM (Gaussian Mixture Model) και έχει προπονηθεί να αναγνωρίζει τα pixel της εικόνας που ανήκουν σε γραμμή του δρόμου. Χρησιμοποιεί τεχνική αντίστοιχη του φίλτρου του Gauss (Υποενότητα 3.2.6), δηλαδή έναν πυρήνα του οποίου το μέγεθος όσο μεγαλύτερο είναι, τόσο καλύτερα απομακρύνει τον θόρυβο από τον δρόμο, δηλαδή τις σκιές (Σχ. 4.5). Έπειτα, εφαρμόζεται ο αλγόριθμος εύρεσης γραμμής RANSAC (Random Sample Consensus), ο οποίος υπολογίζει τα πολυώνυμα που αντιπροσωπεύουν τις γραμμές της λωρίδας. Στη συνέχεια, γίνεται χρήση του φίλτρου Kalman, το

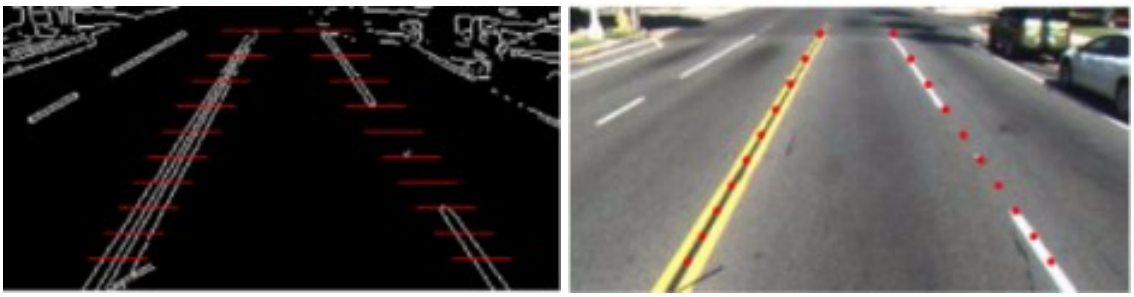


Σχήμα 4.4: Οπτικό αποτέλεσμα της λωρίδας κατά την κίνηση [20]

οποίο εξομαλύνει και βελτιώνει την αναγνώριση των γραμμών (Σχ. 4.6). Τέλος, ακολουθεί μια μέθοδος για την εύρεση του χρώματος και του τύπου της γραμμής, δηλαδή άσπρη ή κίτρινη και μονή, διπλή ή διακεκομμένη. Ο τελικός αλγόριθμος του Xing, δοκιμάστηκε σε διάφορα βιντεοσκοπημένα σενάρια φωτισμού και σκίασης με μεγάλη απόδοση και γρήγορη επεξεργασία της εικόνας και αποτελεί μία εφαρμογή πραγματικού χρόνου.



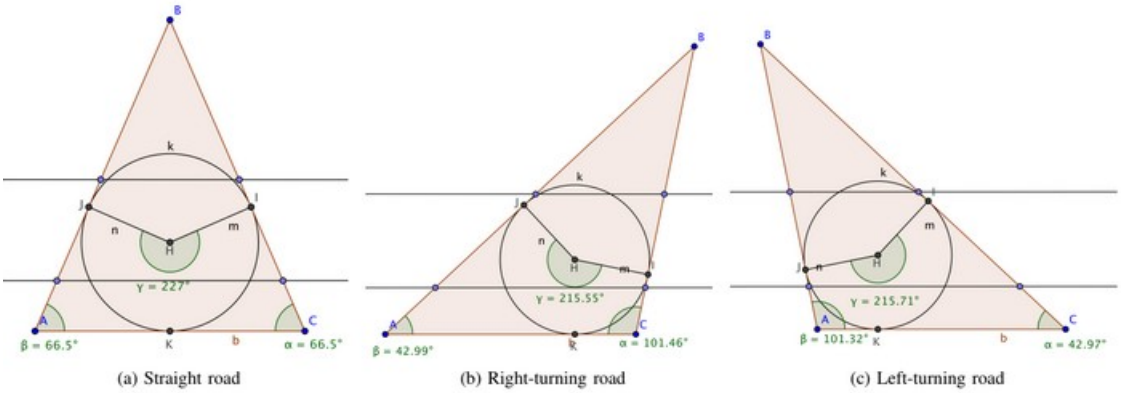
Σχήμα 4.5: Εφαρμογή GMM με μεγαλύτερο μέγεθος πυρήνα για απομάκρυνση σκιών [21]



Σχήμα 4.6: Οι ανίχνευση των σημείων των γραμμών [21]

Με έναν παρόμοιο συνδυασμό με του Xing, προσεγγίζει το πρόβλημα και ο Bottazzi κ.α. [22], με βάση την οπτική και με βάση την ανίχνευση χαρακτηριστικών, ενώ η εύρεση της πορείας γίνεται με τον συνδυασμό των αποτελεσμάτων. Η οπτική προσέγγιση γίνεται με

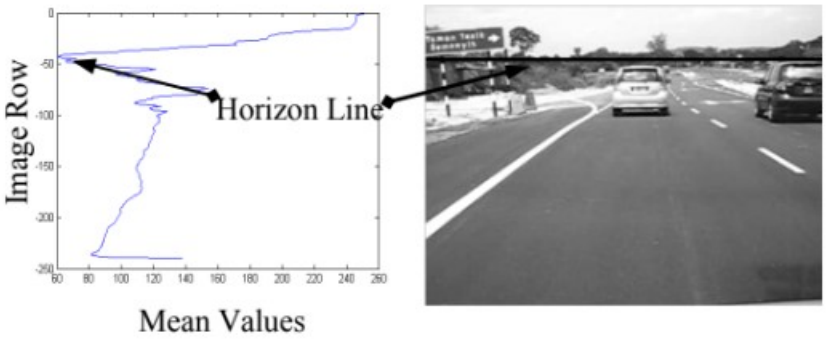
χρήση Gauss-Canny-Hough και εφαρμόζεται σε δύο σταθερά πλαίσια ενδιαφέροντος στην εικόνα, το καθένα από τα οποία ανιχνεύει την δεξιά ή την αριστερή γραμμή. Έπειτα, με αυτό τον τρόπο βρίσκει την απόσταση μεταξύ των γραμμών. Παράλληλα, για την επιβεβαίωση της ύπαρξης των γραμμών, εκτελείται παρακολούθηση χαρακτηριστικών γραμμής (feature tracking), η οποία πραγματοποιείται σε δυναμικά πλαίσια ενδιαφέροντος που μετακινούνται ανάλογα με την προηγούμενη θέση των γραμμών. Για την αποφυγή αναγνώρισης άλλων γραμμών ή του πεζοδρομίου, τα σημεία που είναι υποψήφια σημεία γραμμής δοκιμάζονται με ένα αλγόριθμο τριγωνικού μοντέλου. Τέλος, ο συνδυασμός των αποτελεσμάτων από τους δύο παράλληλους αλγορίθμους εφαρμόζεται επεκτείνοντας τις δύο γραμμές της λωρίδας ώσπου να συναντηθούν στο βάθος του ορίζοντα. Με βάση την γωνία και την απόσταση αυτού του σημείου, μπορεί, στη συνέχεια, να καθοριστεί η πορεία του οχήματος (Σχ. 4.7). Ο αλγόριθμος αυτός δοκιμάστηκε σε αρχεία φωτογραφιών με απαιτητικά σενάρια, όπως συννεφιασμένη μέρα ή σκιές στον δρόμο. Η δοκιμή του είχε μεγάλο ποσοστό επιτυχίας και μπορεί με ευκολία να αναγνωρίσει σωστά την λωρίδα κίνησης, χωρίς την παρεμβολή των διπλανών γραμμών ή του πεζοδρομίου. Η σωστή λειτουργία του, ωστόσο, περιορίζεται σε δρόμους με καλά χρωματισμένες γραμμές.



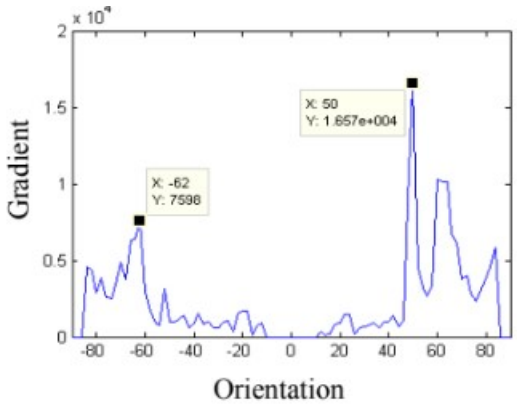
Σχήμα 4.7: Καθορισμός πορείας με βάση το σύστημα προέκτασης [22]

Μια αρκετά διαφορετική προσέγγιση από τις προηγούμενες προτείνει ο Lim κ.α. [23]. Αρχικά με τον υπολογισμό ενός οριζόντιου ιστογράμματος των τιμών pixel της κάθε σειράς της εικόνας, βρίσκει την γραμμή του ορίζοντα και αποκλείει οτιδήποτε είναι πάνω από αυτό το ύψος, δηλαδή τα pixel που αντιστοιχούν στον ουρανό (Σχ. 4.8). Στη συνέχεια, μετατρέπει την υπόλοιπη εικόνα σε ασπρόμαυρη και λαμβάνει την τοπική μέγιστη τιμή των pixel για κάθε γραμμή της εικόνας, και τη συνολική μεγαλύτερη τιμή. Οποιοδήποτε pixel με τιμή μεταξύ της τοπικής μέγιστης και της συνολικής μέγιστης θεωρείται ως πιθανό pixel γραμμής.

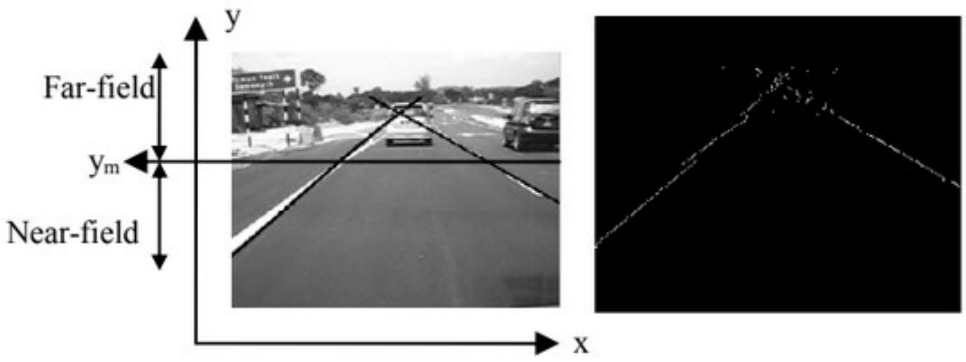
Στη συνέχεια, με μια διαδικασία παρόμοια με αυτή του τελεστή Sobel, υπολογίζεται η κλίση των pixel που προέκυψαν από το προηγούμενο βήμα σε μοίρες. Οι τιμές που βρίσκονται, μεταξύ -90 και 90 μοιρών, αναπαρίστανται σε ένα πίνακα (Σχ. 4.9). Η υψηλότερη θετική τιμή συμβολίζει την κλίση της αριστερής γραμμής, ενώ η υψηλότερη αρνητική την κλίση της δεξιάς γραμμής, δίνοντας, ταυτόχρονα, και την τοποθεσία του κέντρου των γραμμών. Με βάση τα δύο μοντέλα γραμμών που προκύπτουν, ανιχνεύονται τα pixel που είναι πιο κοντά σε αυτές τις γραμμές και θεωρούνται ως πιθανά σημεία γραμμής (Σχ. 4.10). Τέλος, υπολογίζονται δύο παραβολικές καμπύλες βασιζόμενες στα πιθανά σημεία γραμμής, μια για το κοντινό πεδίο (near-field) που συνήθως είναι ευθεία και μία για το μακρινό (far-field) που συνήθως είναι καμπύλη. Η παρούσα λύση είναι αρκετά πρωτοποριακή στη μέθοδο διαχείρισης των δεδομένων για την εύρεση των καμπύλων γραμμών της λωρίδας. Στη δοκιμή της σε βιντεοσκοπημένη δοκιμή είχε αρκετά μεγάλη επιτυχία, ωστόσο, δοκιμάστηκε μόνο σε περιπτώσεις ηλιοφάνειας και καθαρών γραμμών.



Σχήμα 4.8: Εύρεση γραμμής ορίζοντα [23]

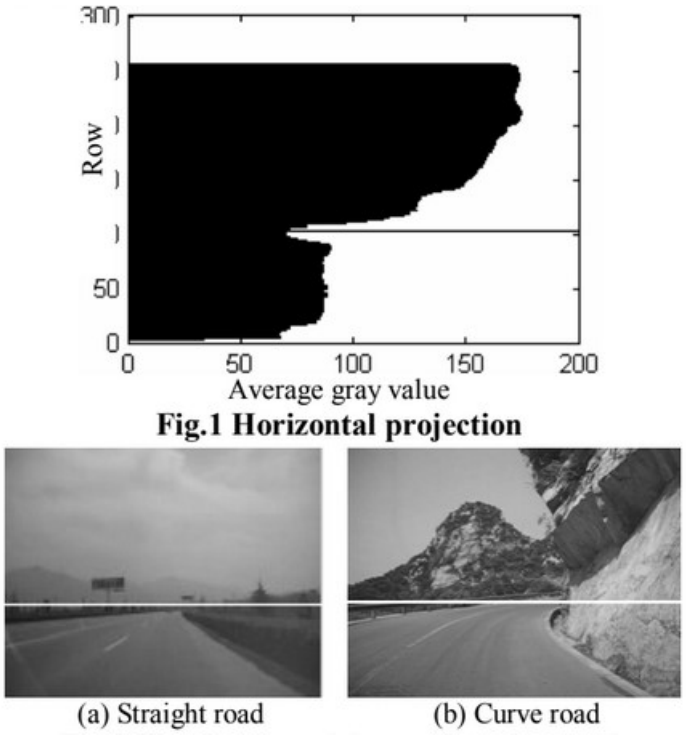


Σχήμα 4.9: Κλίσεις των δύο γραμμών της λωρίδας [23]

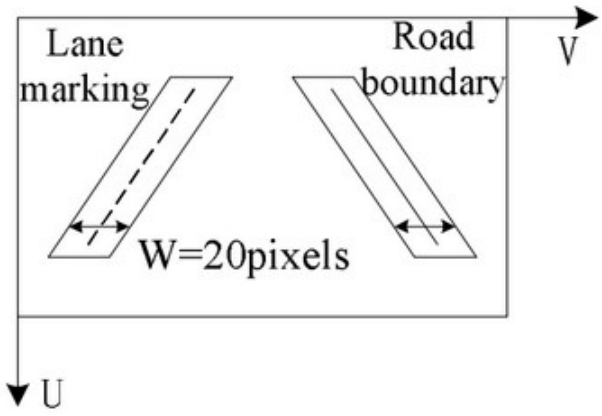


Σχήμα 4.10: Εύρεση των πιθανών σημείων γραμμών [23]

Με παρόμοιο τρόπο με του Lim, ξεκινάει και ο αλγόριθμος του Lu κ.α. [24], δηλαδή με τον υπολογισμό ενός οριζόντιου ιστογράμματος των γραμμών pixel της ασπρόμαυρης εικόνας, με στόχο την εύρεση της γραμμής του ορίζοντα (Σχ. 4.11). Αυτό γίνεται για να μην λαμβάνονται υπόψιν στην επεξεργασία τα pixel που βρίσκονται πάνω από αυτήν τη γραμμή. Έπειτα, με τον υπολογισμό ενός κάθετου ιστογράμματος στις στήλες της εικόνας και με την εφαρμογή ενός ειδικού αλγόριθμου διαχωρισμού που βασίζεται στον υπολογισμό της διασποράς των μέσων τιμών των pixel, τα pixel κατανέμονται σε δύο κατηγορίες, τα pixel γραμμής και τα υπόλοιπα. Εν συνεχεία, με την εφαρμογή του τελεστή Sobel πάνω στην εικόνα, και τον υπολογισμό των διαφορών μεταξύ γειτονικών pixel, υπολογίζονται τα χαρακτηριστικά pixel των γραμμών. Τέλος, με την μέθοδο των ελαχίστων τετραγώνων υπολογίζονται τα μοντέλα γραμμών που ταιριάζουν καλύτερα στις γραμμές της εικόνας. Συνήθως, χρειάζεται μια απλή ευθεία για να εκφράσει την κάθε γραμμή της λωρίδας. Αν η κλίση της λωρίδας είναι απότομη, η αναπαράστασή της γίνεται με μία καμπύλη γραμμή που την προσεγγίζει. Ένα χαρακτηριστικό που κάνει αυτόν τον αλγόριθμο να ξεχωρίζει, είναι πως οι παραπάνω υπολογισμοί γίνονται μία φορά ανά 20 frame για τα pixel που βρίσκονται κάτω από την γραμμή του ορίζοντα. Για τα υπόλοιπα 19, η αναζήτηση γραμμών περιορίζεται μόνο μέσα σε δύο μικρά πεδία που καλύπτουν τον χώρο στον οποίο εμφανίζονταν οι γραμμές στο προηγούμενο frame. Η λογική απορρέει από το γεγονός ότι η γραμμή στο επόμενο frame δεν μπορεί να έχει μεγάλη απόκλιση στην τοποθεσία, οπότε θα βρίσκεται μέσα στο όρια των 20 pixel από το προηγούμενο frame (Σχ. 4.12). Με αυτόν τον τρόπο η επεξεργασία των υπόλοιπων frame γίνεται πολύ πιο γρήγορα και καθιστά τον αλγόριθμο κατάλληλο για εφαρμογές πραγματικού χρόνου. Ωστόσο, όπως και οι προηγούμενες, εφαρμόστηκε μόνο σε βιντεοσκοπημένες διαδρομές.



Σχήμα 4.11: Υπολογισμός της γραμμής του ορίζοντα [24]



Σχήμα 4.12: Εύρεση των πιθανών νέων γραμμών μέσα στα πλαίσια που προκύπτιυν από τις γραμμές του προηγούμενου frame [24]

Κεφάλαιο 5

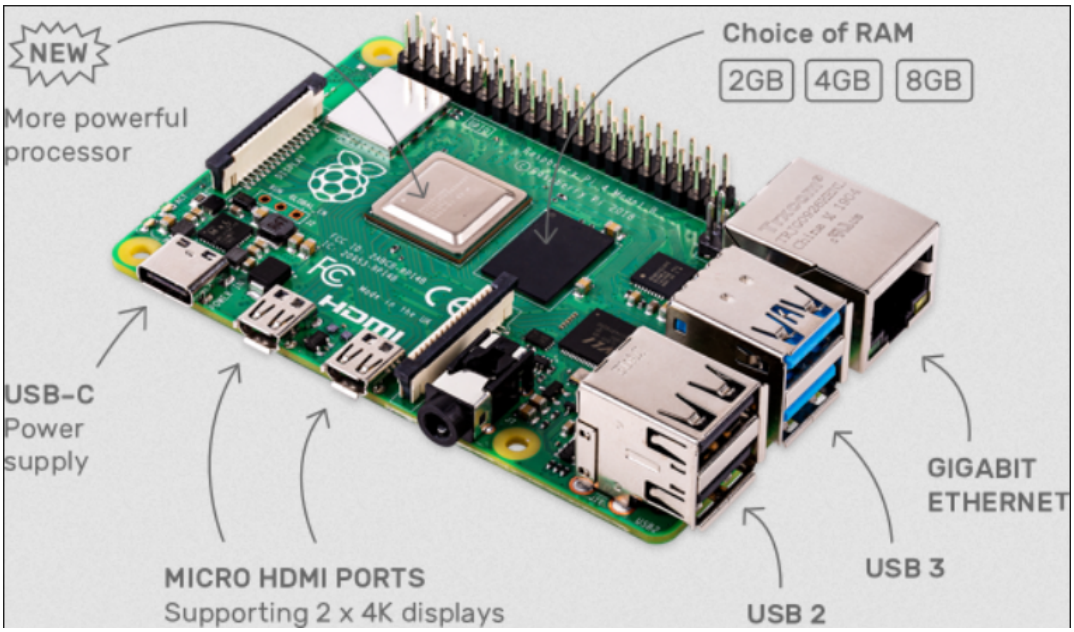
Hardware - Μεθοδολογία Κατασκευής Οχήματος

5.1 Εξαρτήματα και μηχανισμοί που χρησιμοποιήθηκαν

5.1.1 Πλατφόρμα συστήματος - Raspberry Pi

Για τον κεντρικό έλεγχο του συστήματος του οχήματος και του ρομποτικού βραχίωνα χρησιμοποιήθηκε ένα Raspberry Pi. Είναι ένας, μικρού μεγέθους, υπολογιστής με χαμηλή κατανάλωση και χαμηλό κόστος. Οι αποδόσεις του, είναι παρόμοιες με αυτές ενός συνηθισμένου σταθερού ή φορητού υπολογιστή. Διαφοροποιείται, όμως, δίνοντας περαιτέρω επιλογές λόγω των κεφαλών σύνδεσης γενικής χρήσης (General Purpose Input/Output Pins [GPIO Pins]) και των πολλών Hardware on Top (HAT). Τα HAT μπορούν να συνδεθούν στο Raspberry Pi επεκτείνοντας τις δυνατότητες και τις επιλογές που προσφέρει, πέρα από τις βασικές χρήσεις ενός οικιακού υπολογιστή.

Το μοντέλο που επιλέχθηκε, ανάμεσα στις πολλές εκδοχές που είναι διαθέσιμες είναι το πιο σύγχρονο Raspberry Pi 4 Model B+ της έκδοσης των 4GB RAM (Σχ. 5.1).



Σχήμα 5.1: Raspberry Pi 4 Model B [25]

Οι προδιαγραφές του είναι αυτές που διαφαίνονται στο Σχ. 5.2:

Processor:
Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz

Memory:
1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC

Connectivity:
2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE
Gigabit Ethernet
2 Ã— USB 3.0 ports
2 Ã— USB 2.0 ports.

GPIO:
Standard 40-pin GPIO header (fully backwards-compatible with previous boards)

Video & sound:
2 Ã— micro HDMI ports (up to 4Kp60 supported)
2-lane MIPI DSI display port
2-lane MIPI CSI camera port
4-pole stereo audio and composite video port

Multimedia:
H.265 (4Kp60 decode);
H.264 (1080p60 decode, 1080p30 encode);
OpenGL ES, 3.0 graphics

SD card support:
Micro SD card slot for loading operating system and data storage

Input power:
5V DC via USB-C connector (minimum 3A1)
5V DC via GPIO header (minimum 3A1)
Power over Ethernet (PoE)"enabled (requires separate PoE HAT)

Environment:
Operating temperature 0-50°C

Σχήμα 5.2: Specifications of Raspberry Pi 4 Model B [25]

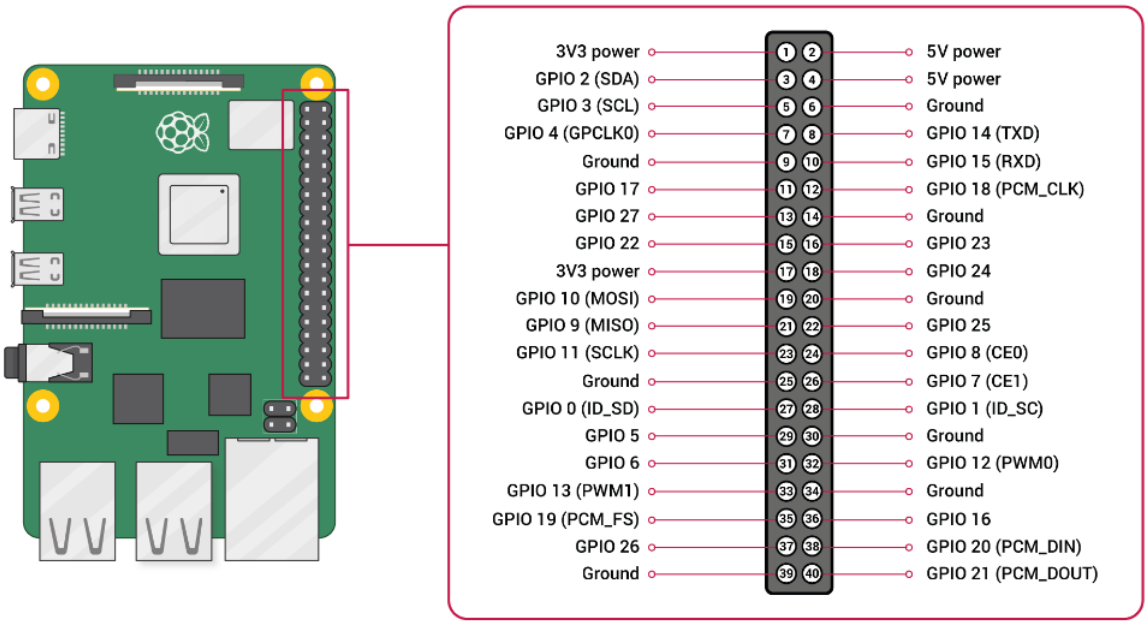
Τέλος, κάποιες από τις συνήθεις χρήσεις του Raspberry Pi είναι:

1. Έλεγχος ρομποτικών συσκευών
2. Φιλοξενία Ιστοσελίδων και διαδικτυακών Εφαρμογών
3. Δημιουργία Οικιακού Διαδικτύου Πραγμάτων (Internet of Things [IoT])
4. Απλή οικιακή χρήση ως αντικαταστάτης ενός συνηθισμένου υπολογιστή
5. Κέντρο ψυχαγωγίας με απευθείας σύνδεση σε τηλεόραση και ηχεία
6. Κονσόλα για ηλεκτρονικά παιχνίδια παλαιότερης εποχής (Ρετρό)
7. Διακομιστής (Server) παιχνιδιών πολλών παικτών
8. Κέντρο Παρακολούθησης Οικιακού Δικτύου και Διακοπής Διαφημίσεων (Advertise Blocker) κ.α.

Η επιλογή του Raspberry Pi αποτέλεσε μονόδρομο για την επιτυχή αυτόνομη λειτουργία του οχήματος (μετακίνηση και λειτουργία της βάσης και του ρομποτικού βραχίωνα). Ταυτόχρονα έδωσε τη δυνατότητα της εκτέλεσης προγραμμάτων πραγματικού χρόνου, όπως η αναγνώριση και η παρακολούθηση λωρίδας. Εναλλακτικά, θα ήταν απαραίτητη η χρήση κάποιου μικροελεγκτή που θα έστελνε την πληροφορία από την κάμερα σε έναν άλλο υπολογιστή μέσω δικτύου, να την επεξεργαστεί και να επιστρέψει τις ανάλογες εντολές για περιστροφή του βραχίωνα ή μετακίνηση του οχήματος. Η εναλλακτική αυτή, όμως, θα δημιουργούσε προβλήματα με την ταχύτητα επεξεργασίας του κάθε στιγμιότυπου εικόνας (frame), καθώς έτσι προστίθεται στην διάρκεια της επεξεργασίας και όλη η μεταφορά της πληροφορίας εικόνας και των εντολών κίνησης μέσα από το οικιακό δίκτυο.

5.1.2 Κεφαλές Σύνδεσης Γενικής Χρήσης - General Purpose Input/Output Pins (GPIO)

Η επιλογή χρήσης του Raspberry Pi βασίστηκε κατά κύριο λόγο στα πλεονεκτήματα που προσφέρουν οι GPIO κεφαλές του (Σχ. 5.3). Οι 40 κεφαλές που διαθέτει, δίνουν ποικίλες επιλογές αισθητήρων και ηλεκτρονικών συσκευών για σύνδεση, που αποτελούν την πηγή πληροφοριών για το εξωτερικό περιβάλλον.



Σχήμα 5.3: Οι κεφαλές του Raspberry Pi 4 Model B [25]

Ορισμένες κεφαλές GPIO δίνουν την δυνατότητα αποστολής σήματος με διαμόρφωση πλάτους-παλμού (Pulse-Width Modulation[PWM]). Αυτές παρέχουν ευκολία στον έλεγχο των κινητήρων Servo, οι οποίοι χρησιμοποιούνται για την κίνηση των αρθρώσεων και της δαγκάνας του ρομποτικού βραχίωνα, καθώς και για τον έλεγχο της λειτουργίας των κινητήρων των τροχών του οχήματος για την μετακίνησή της βάσης. Τρεις από τις κεφαλές GPIO του Raspberry Pi μπορούν να χρησιμοποιηθούν και για την απευθείας τροφοδότηση των αισθητήρων και των συσκευών με τάση 5V ή 3V. Σε αυτή τη περίπτωση μειώνεται η απόδοση επεξεργασίας του συστήματος σε απαιτητικές εφαρμογές πραγματικού χρόνου, οπότε προτιμάται η χρήση εξωτερικής τροφοδοσίας.

5.1.3 Κινητήρες Servo

Οι κινητήρες που χρησιμοποιήθηκαν για την λειτουργία των κινήσεων του ρομποτικού βραχίωνα ονομάζονται κινητήρες Servo. Αυτοί είναι ηλεκτρομηχανικοί κινητήρες, οι οποίοι εξαρτώμενοι από την παροχή τάσης και ρεύματος, παράγουν την ανάλογη ροπή και ταχύτητα και επιτρέπουν τον ακριβή έλεγχο της γωνιακής τους θέσης. Η λειτουργία τους βασίζεται στην ανταλλαγή πληροφοριών με έναν ελεγκτή, ο οποίος ελέγχει την κίνηση του Servo με βάση τις πληροφορίες που λαμβάνει για τη θέση του, από έναν αισθητήρα ανάδρασης που είναι ενσωματωμένος στο Servo. Τα είδη Servo ποικίλουν και εφαρμόζονται σε διάφορους

τομείς, όπως στη ρομποτική, σε μηχανήματα κοπής ακριβείας CNC (Computer Numerical Control) και σε αυτοματοποιημένες κατασκευές. Το σήμα εισόδου του Servo μεταφράζεται στην επιθυμητή θέση του άξονα, η οποία ελέγχεται με βάση το σύστημα ανατροφοδότησης. Η θέση του άξονα, που υπολογίζεται με τη χρήση συνήθως ενός ποτενσιόμετρου, συγκρίνεται με την απαιτούμενη θέση. Εάν είναι διαφορετική τότε παράγεται ένα σήμα σφάλματος που προκαλεί την κίνηση του άξονα προς την επιθυμητή θέση. Το σήμα σφάλματος σταματάει όταν ο άξονας φτάσει ακριβώς στην θέση του [45].

Στην κατασκευή του οχήματος χρησιμοποιήθηκε το μοντέλο MG 995 (Metal Gear 995) (Σχ. 5.4), το οποίο διαθέτει μεταλλικό εσωτερικό μηχανισμό που του επιτρέπει να είναι πιο σταθερό και ισορροπημένο, ενώ ταυτόχρονα υποστηρίζει μεγαλύτερο βάρος. Λειτουργεί με τάση 4.8V ή 6V και παράγει μέτρο ροπής ίσο με 13 και 15 kgcm, αντίστοιχα. Δηλαδή, όση ροπή θα προκαλούσε μία μάζα βάρους 13 ή 15 kg σε απόσταση 1 cm από το σημείο μέτρησης. Η ροπή που παράγει είναι και ο λόγος της επιλογής του συγκεκριμένου κινητήρα, καθώς θα μπορεί να κινεί με ευκολία το βάρος του βραχίωνα. Το εύρος κίνησης του είναι 180 μοίρες ή, αλλιώς, μισή περιστροφή. Διαθέτει τρία καλώδια σύνδεσης, δύο για την παροχή συνεχούς ρεύματος (θετικό και αρνητικό) και ένα καλώδιο για το σήμα ελέγχου. Ο έλεγχος του γίνεται με σήματα PWM μέσω σύνδεσης με μία GPIO θύρα του Raspberry Pi.

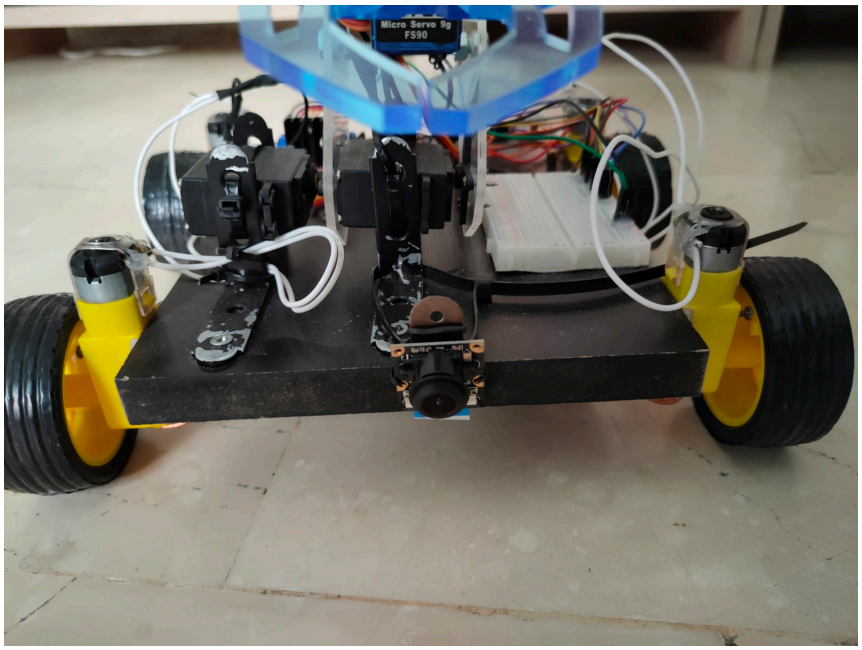
Σχήμα 5.4: Το servo MG995 [26]

5.1.4 Pi Camera

Ως βασικός αισθητήρας του οχήματος για την αντίληψη του εξωτερικού περιβάλλοντος επιλέχθηκε μια κάμερα, με στόχο να προσομοιάζεται ο τρόπος που λειτουργούν τα περισσότερα αυτόνομα αυτοκίνητα της βιομηχανίας. Εν αντιθέσει με τα προηγούμενα, ορισμένα οχήματα χρησιμοποιούν αντί για κάμερα, αισθητήρες LiDAR οι οποίοι, ωστόσο, έχουν μεγάλο κόστος.

Αρχικά, είχε χρησιμοποιηθεί μια απλή διαδικτυακή κάμερα (web camera), από την οποία είχε αφαιρεθεί το περίβλημα για καλύτερη και σταθερότερη τοποθέτηση. Η σύνδεση της γινόταν με χρήση USB 2.0. Ωστόσο, λόγω του γεγονότος ότι ο φακός δεν ήταν ευρυγώνιος, η κάμερα δεν παρείχε σωστή οπτική των γραμμών οπότε δεν λειτουργούσε σωστά ο αλγόριθμος. Εν τέλει, αντικαταστάθηκε η προηγούμενη κάμερα από την Raspberry Pi Camera with fishlens (Σχ. 5.5 και 5.6), μια κάμερα με ευρυγώνιο φακό και απευθείας ειδική σύνδεση με το raspberry pi. Η αλλαγή αυτή επέφερε καλύτερη οπτικοποίηση των γραμμών και ταχύτερη μετάδοση στο raspberry σε σχέση με την usb camera, η οποία εμφάνιζε μια μικρή αλλά ωστόσο σημαντική καθυστέρηση στην μετάδοση. Επιπλέον, διευκόλυνε την επεξεργασία της πληροφορίας της εικόνας χάρη στην ειδική βιβλιοθήκη εντολών που υπάρχει για αυτή την κάμερα.

Σχήμα 5.5: Ευρυγώνεια κάμερα για Raspberry Pi [27].

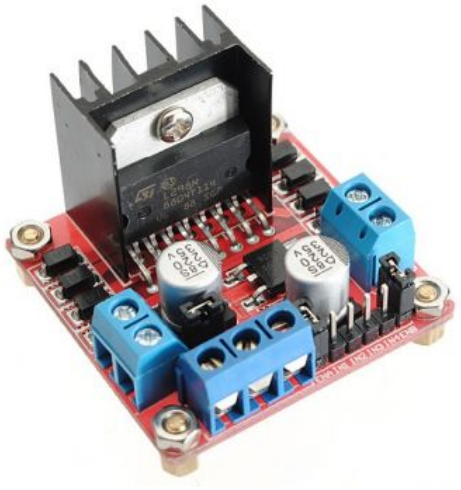


Σχήμα 5.6: Τοποθέτηση της κάμερας στο όχημα

5.1.5 Μετακίνηση του οχήματος

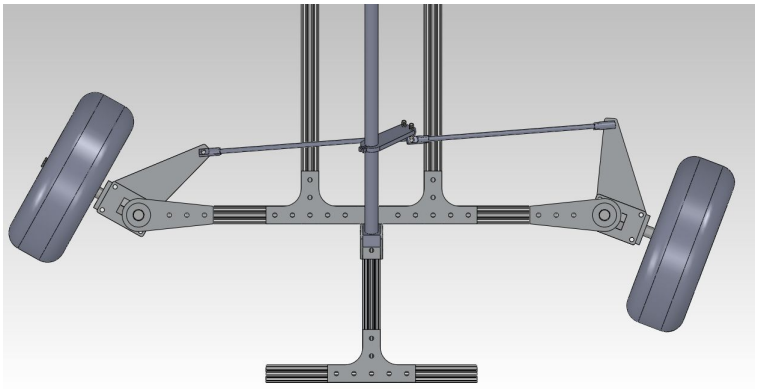
Για την μετακίνηση της βάσης του οχήματος χρησιμοποιήθηκαν τέσσερις κινητήρες συνεχούς ρεύματος με τάση λειτουργίας στα 3V (Σχ. 5.7). Για τον έλεγχο των κινητήρων χρησιμοποιήθηκε ένας ελεγκτής (Motor Controller) με ενσωματωμένο chip L298N (Σχ. 5.8), ο οποίος συνδέεται με το raspberry pi και δίνει την δυνατότητα ελέγχου της ταχύτητας και της κατεύθυνσης δύο ανεξάρτητων κινητήρων. Η τροφοδοσία του γίνεται από εξωτερική πηγή για να μην επιβαρύνεται το Raspberry.

Σχήμα 5.7: Κινητήρας συνεχούς ρεύματος [27]



Σχήμα 5.8: Motor Controller with L298N Chip [27]

Θεωρητικά, η καλύτερη επιλογή για την πηδαλιούχηση του οχήματός θα ήταν η χρήση μηχανισμού παρόμοιου με αυτού των αυτοκινήτων (Σχ. 5.9) που προσφέρει τη δυνατότητα ελέγχου της επιθυμητής διεύθυνσης των τροχών. Ωστόσο, θα χρειαζόταν η προμήθεια ειδικού μηχανισμού που θα έπρεπε να τοποθετηθεί σε εξειδικευμένο αμάξωμα, πράγμα που δεν ήταν εφικτό σε πρακτικό επίπεδο με τους πόρους του εγχειρήματος.



Σχήμα 5.9: Μηχανισμός πηδαλιούχησης [28]

5.1.6 Δαγκάνα του Ρομποτικού Βραχίωνα

Το βασικό τμήμα για την χρήση ενός ρομποτικού βραχίωνα είναι η δαγκάνα στην άκρη του. Με αυτήν μπορεί να επιτελέσει διάφορες εργασίες στις οποίες απαιτείται να πιάσει και να αφήσει κάποιο αντικείμενο. Χρησιμοποιήθηκε η δαγκάνα του Σχ. 5.10, η οποία αποτελείται από συναρμολογούμενα κομμάτια. Η κίνηση της, δηλαδή άνοιγμα και κλείσιμο, ελέγχεται από ένα κινητήρα Servo μικρότερων διαστάσεων και ροπής από αυτούς που χρησιμοποιή-

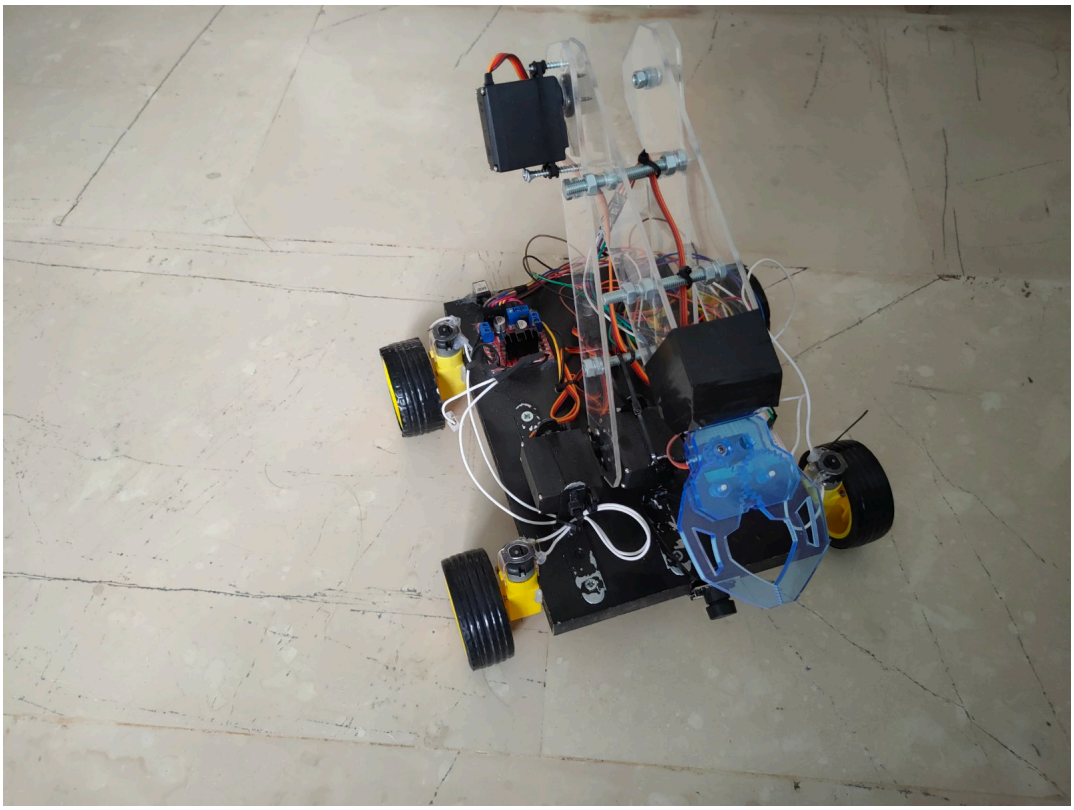
θηκαν για τις αρθρώσεις του βραχίωνα, ο οποίος όμως ελέγχεται με τον ίδιο τρόπο.

Σχήμα 5.10: Δαγκάνα του ρομποτικού βραχίωνα που ελέγχετι από κινητήρα Servo [27]

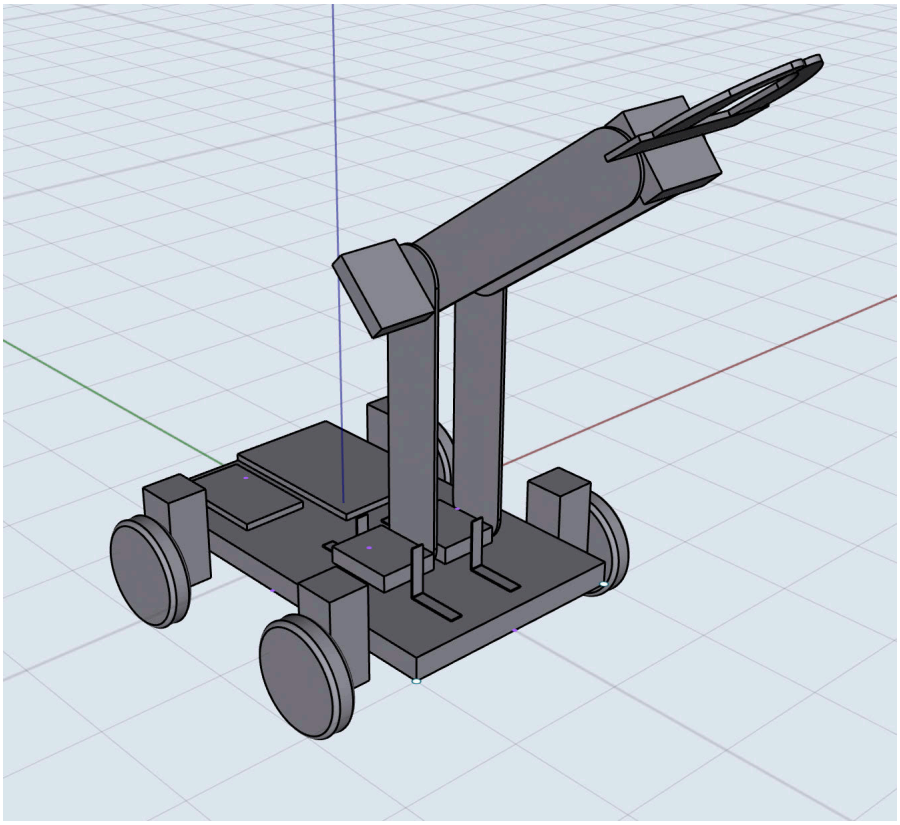
5.2 Συνολική Κατασκευή

5.2.1 Βάση του οχήματος

Στη βάση του οχήματος είναι τοποθετημένοι οι τέσσερις κινητήρες συνεχούς ρεύματος (Σχ. 5.11 και 5.12). Επιλέχθηκε η τετρακίνηση λόγω του μεγάλου συνολικού βάρους του οχήματος, καθώς όταν δοκιμάστηκε η λειτουργία του με τους δύο μόνο κινητήρες να δίνουν κίνηση υπήρχε δυσκολία στους ελιγμούς περιστροφής και στην λειτουργία χαμηλής ταχύτητας. Οι κινητήρες λειτουργούν σε ζευγάρια με ίδιες κινήσεις στους δύο δεξιά και στους δύο αριστερά. Η τροφοδοσία τους, θετικό και αρνητικό, γίνεται μέσω του Motor Controller, που είναι τοποθετημένο στο οπίσθιο μέρος της βάσης του οχήματος δίπλα από το raspberry pi, και συνδέεται απευθείας με εξωτερικό μετασχηματιστή συνεχούς ρεύματος. Ο έλεγχος για την κίνηση των κινητήρων δίνεται από τις GPIO κεφαλές του Raspberry pi, με τις οποίες είναι απευθείας συνδεδεμένο.



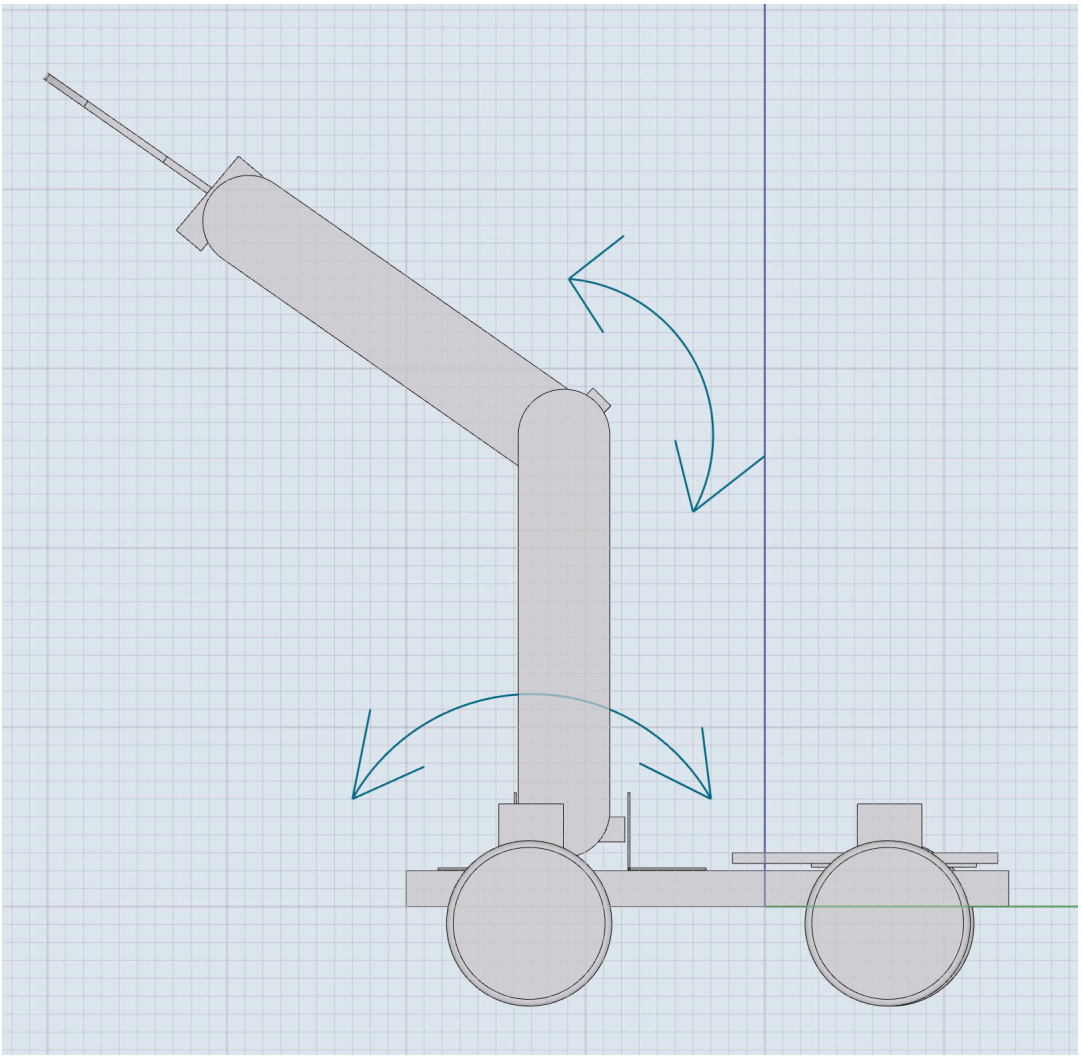
Σχήμα 5.11: Η βάση του οχήματος



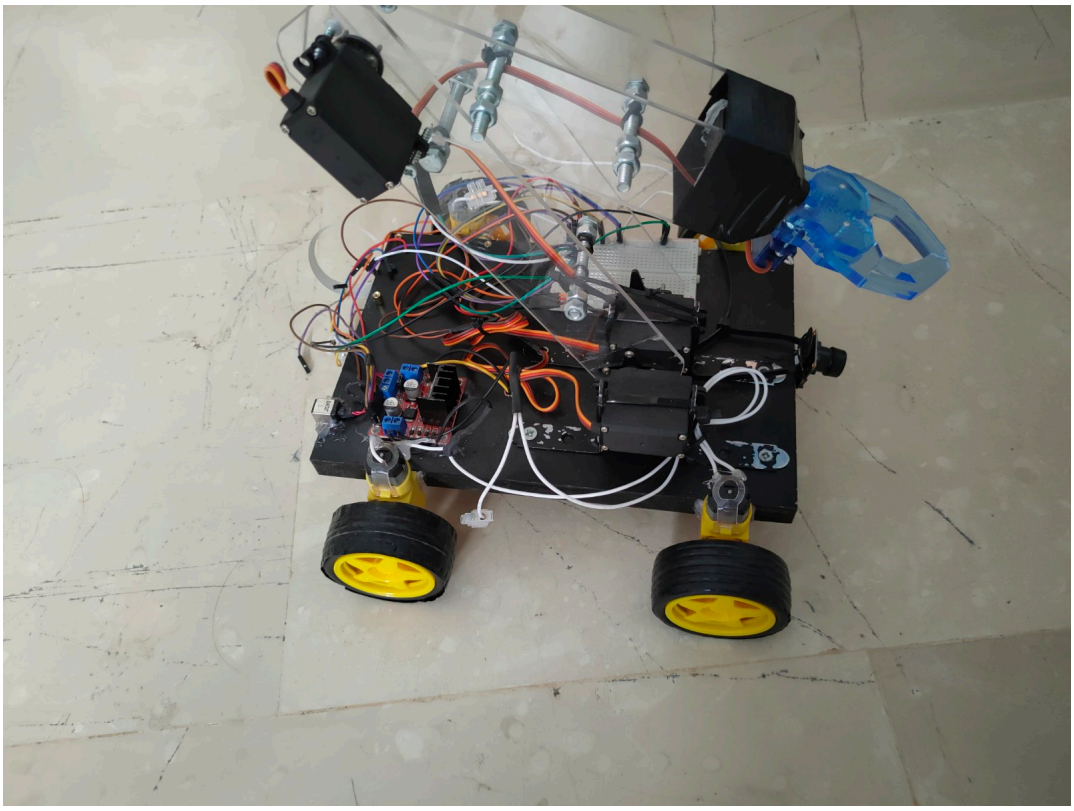
Σχήμα 5.12: Τρισδιάτατη οπτική του οχήματος

5.2.2 Ρομποτικός Βραχίωνας

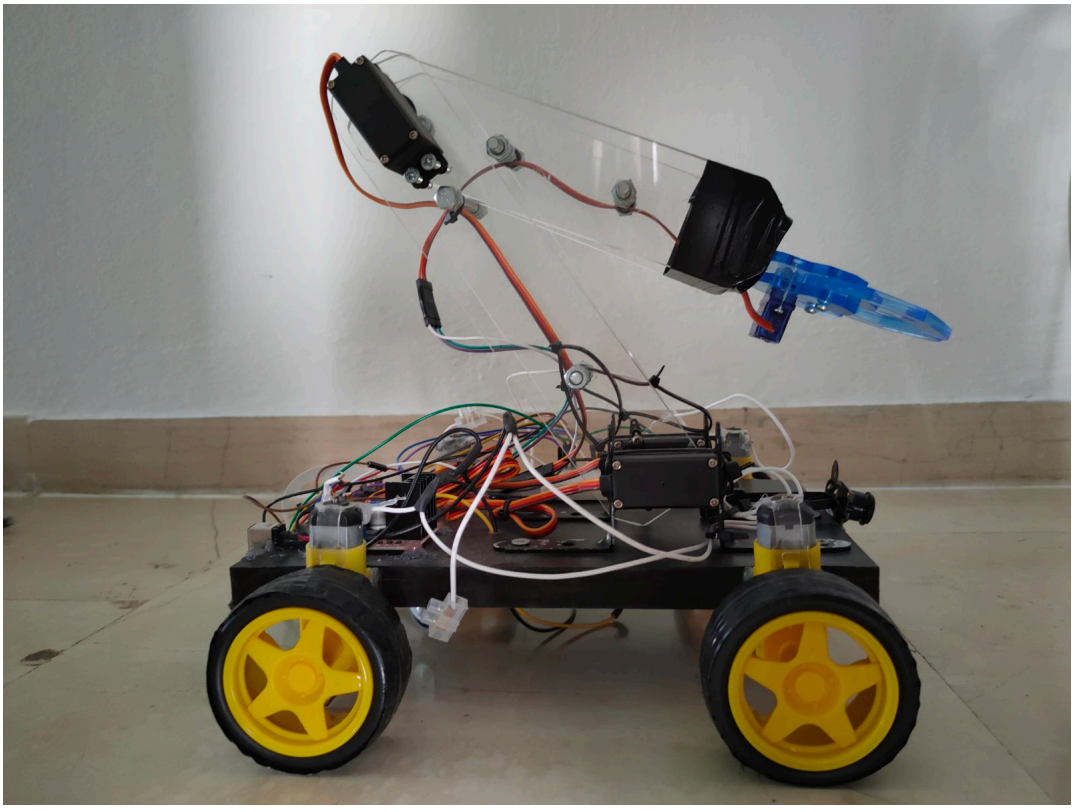
Ο ρομποτικός βραχίωνας έχει δύο βαθμούς ελευθερίας, δηλαδή δύο αρθρώσεις. Η πρώτη βρίσκεται στη βάση και η δεύτερη στην μέση της έκτασης. Σε συνδυασμό με την δυνατότητα περιστροφής του οχήματος, έχει συνολικά τρεις βαθμούς ελευθερίας (Σχ. 5.13). Ο ρομποτικός βραχίωνας κατασκευάστηκε εξ' ολοκλήρου από ακρυλικό γυαλί (plexiglass). Απαρτίζεται από δύο ζευγάρια κομματιών plexiglass (στο σύνολο τέσσερα), που διατηρούνται παράλληλα ενωμένα με βίδες, ώστε να υπάρχει καλή ισορροπία μεταξύ των κινήσεων. Για την άρθρωση στη βάση υπάρχουν δύο servo, λόγω της μεγάλης ροπής που ασκείται από το βάρος του βραχίωνα όταν είναι σε πλήρη έκταση προς τα μπροστά, ενώ η δεύτερη άρθρωση στο μέσο της έκτασης του βραχίωνα λειτουργεί βασισμένη σε ένα servo (Σχ. 5.14). Τέλος, πάνω στην άκρη του ρομποτικού βραχίωνα έχει στερεωθεί η δαγκάνα (Σχ. 5.15).



Σχήμα 5.13: Οι βαθμοί ελευθερίας του βραχίωνα



Σχήμα 5.14: Τα ζευγάρια plexiglass



Σχήμα 5.15: Η πλαϊνή πλευρά του οχήματος

Κεφάλαιο 6

Software - Λογισμικό για την αυτόνομη οδήγηση του οχήματος

6.1 Εισαγωγή

Στο κεφάλαιο αυτό αναλύεται και επεξηγείται ο κώδικας που χρησιμοποιήθηκε για την επιτυχή αυτόνομη κίνηση του οχήματος πάνω σε μία αυτοσχέδια λωρίδα. Ο σύνδεσμος του Google Drive που περιέχει τον κώδικα και τα βίντεο της αυτόνομης οδήγησης του οχήματος ειναι ο ακόλουθος [46]. Για την καλύτερη εξοικείωση με το ζήτημα της αναγνώρισης γραμμών υλοποιήθηκε, αρχικά, αλγόριθμος για την ακολουθία μίας μόνο γραμμής. Ο αλγόριθμος αυτός υπάρχει και εξηγείται στο Παράρτημα B. Η επιλογή να εφαρμοστεί πρώτα ο αλγόριθμος για μία γραμμή έγινε γιατί αποτελεί λιγότερο περίπλοκο πρόβλημα σε σχέση με την ακολουθία μιας λωρίδας, που απαρτίζεται από δύο γραμμές. Οι στόχοι ήταν:

1. Να απομονωθούν τα pixel που ανήκουν στην γραμμή.
2. Να υπολογιστεί η κλίση και η θέση της γραμμής
3. Μετά από δοκιμές να επιτευχθεί ο συντονισμός της οδήγησης της διαδρομής ανάλογα με την κλίση της στροφής ή την διόρθωση της απόστασης από την γραμμή σε περίπτωση απόκλισης από αυτή.

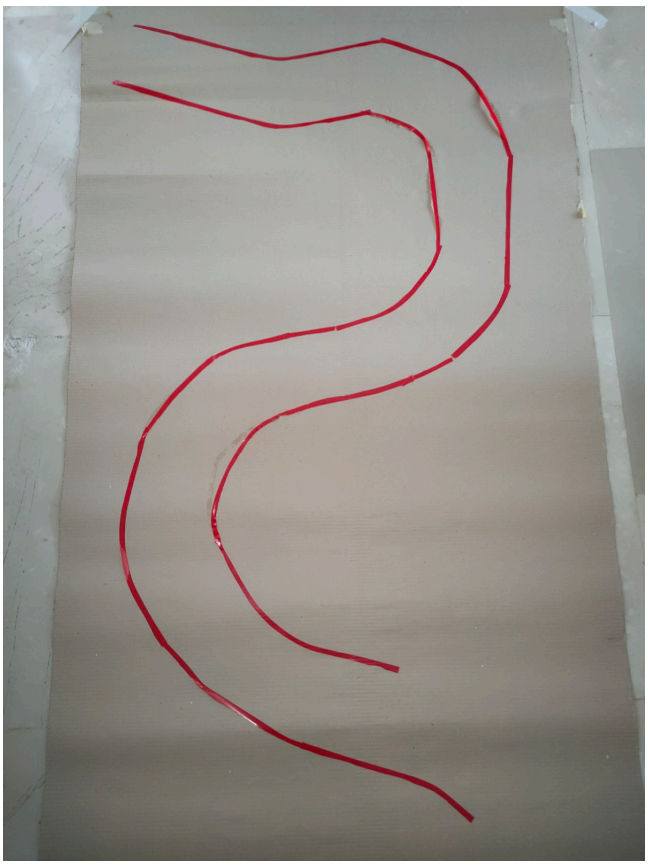
Με βάση τα αποτελέσματα του αρχικού αλγορίθμου και την εμπειρία που αποκτήθηκε κατά την εφαρμογή του, δόθηκε η δυνατότητα για περαιτέρω ανάπτυξη και εξέλιξη του σε αλγόριθμο ανίχνευσης και ακολουθίας δύο γραμμών μιας λωρίδας κίνησης.

Ερευνήθηκαν διάφοροι τρόποι ακολουθίας της γραμμής, οι οποίοι δοκιμάστηκαν πρώτα στην αναγνώριση μίας γραμμής και έπειτα για την αναγνώριση μιας λωρίδας. Η τελική επιλογή αποτέλεσε συνδυασμό των αλγορίθμων Canny και Hough. Οι λόγοι που επιλέχθηκε αυτός ο συνδυασμός ποικίλουν. Αρχικά, λόγω της χρήσης raspberry pi στη παρούσα εργασία δεν ήταν εφικτή η χρήση πολυπλοκότερων αλγορίθμων, καθώς θα υπήρχε μείωση της ταχύτητας επεξεργασίας, δυσκολεύοντας την εφαρμογή σε πραγματικό χρόνο. Αυτό οφείλεται στο γεγονός ότι πολλές από τις έρευνες που εφαρμόζουν αυτούς τους αλγορίθμους χρησιμοποιούν επεξεργαστές μεγαλύτερης ισχύος.

Μία καλή εναλλακτική θα ήταν η χρήση του αλγορίθμου Συρόμενων Παραθύρων. Ωστόσο, λόγω του μικρού μεγέθους του συνολικού πρότζεκτ, δηλαδή του οχήματος και της πίστας, δεν θα είχε το επιθυμητό αποτέλεσμα καθώς οι στροφές που χρησιμοποιήθηκαν στην πίστα είναι απότομες. Η τεχνική των συρόμενων παραθύρων βασίζεται στον μετασχηματισμό προοπτικής, οπότε σε αυτή την περίπτωση, λόγω του μετασχηματισμού, θα αντιλαμβανόταν μικρό κομμάτι της στροφής και άρα δεν θα υπήρχε περίπτωση βελτίωσης του αλγορίθμου. Επιπλέον, η χρήση του αλγορίθμου αυτού σε άλλες έρευνες έγινε σε εικόνες που λήφθηκαν μέσα από ένα κανονικού μεγέθους όχημα σε ένα κανονικό δρόμο που ήταν κυρίως ευθεία με μικρές κλίσεις στις στροφές. Εκεί ο αλγόριθμος των Συρόμενων Παραθύρων δουλεύει άψογα καθώς βασίζεται στην λωρίδα που εκτείνεται προς το βάθος της εικόνας και υπολογίζει τις καμπύλες των δύο γραμμών. Αντιθέτως, στο παρόν εγχείρημα δεν υφίσταται βάθος στην λωρίδα, καθώς οι διαστάσεις είναι πολύ μικρότερες και η διαδρομή δεν είναι σε έναν περιφερειακό δρόμο αλλά σε μια στενή και με απότομες στροφές διαδρομή.

Θεωρητικά, για την βέλτιστη απόδοση του συστήματος και την προσομοίωση του δρόμου, θα ήταν χρήσιμη μία ειδική πίστα ασφάλτου με χρωματισμένες τις γραμμές της λωρίδας. Ωστόσο, λόγω έλλειψης των ζητούμενων εγκαταστάσεων ακολουθήθηκαν άλλες μέθοδοι. Αρχικά, δοκιμάστηκε η αναγνώρισης μαύρης γραμμής από μαύρη κολλητική ταινία που είχε τοποθετηθεί στο δάπεδο. Το αποτέλεσμα, όμως, απείχε από το ικανοποιητικό, καθώς υπήρχαν προβλήματα που προέκυπταν από τον φωτισμό του χώρου. Ακόμη, την εφαρμογή του αλγορίθμου εμπόδιζαν οι αυλακώσεις και τα σημάδια από τα πλακάκια στο δάπεδο. Εν τέλει, επιλέχθηκε η χρήση κόκκινης ταινίας, μιας και αυτό το χρώμα δεν υπήρχε στο δάπεδο και στον περιβάλλοντα χώρο. Η κόκκινη κολλητική ταινία εφαρμόστηκε σε ένα ρολό χαρτόνι μεγάλου πάχους (Σχ. 6.1) με σκοπό την αποφυγή των αυλακώσεων και του αντικατοπτρισμού του φωτισμού στο έδαφος. Δύο ακόμη λόγοι της εφαρμογής των παραπάνω αποτελεί η ευκο-

λία μετακίνησης της πίστας και η καλύτερη τριβή των τροχών σε σύγκριση με το ολισθηρό δάπεδο.



Σχήμα 6.1: Η διαδρομή που κατασκευάστηκε

6.2 Κώδικας κίνησης τροχών - Motor Module

Το αρχείο κώδικα python, το οποίο είναι υπεύθυνο για τον έλεγχο των κινητήρων κίνησης του οχήματος, χρησιμοποιείται ως ανεξάρτητο αρχείο, και ονομάζεται MotorModule.py (Κώδ. 6.1). Στην ουσία, δημιουργεί μια κλάση Κινητήρα, η οποία όταν καλείται από άλλα αρχεία κώδικα, δημιουργεί ένα αντικείμενο που προσδιορίζει τους κινητήρες.

Αρχικά, ορίζονται με τις εντολές import της python οι βιβλιοθήκες που χρησιμοποιούνται στον κώδικα, όπως, η βιβλιοθήκη ελέγχου των θυρών GPIO και η εντολή sleep από την βιβλιοθήκη time που διακόπτει την εκτέλεση του προγράμματος για τον ορισμένο χρόνο που δόθηκε κατά την κλήση της.

Στη συνέχεια, καλείται η συνάρτηση init, η οποία αρχικοποιεί το νέο αντικείμενο κλάσης Motor που δημιουργείται. Κατά την κλήση της ορίζονται ποιες είναι οι κεφαλές GPIO που έχουν χρησιμοποιηθεί για την σύνδεση με το τσιπ ελέγχου των κινητήρων. Τέλος, ορίζεται η

συνάρτηση move,η οποία καλείται από το βασικό πρόγραμμα όταν θέλει να πραγματοποιηθεί κίνηση του οχήματος. Ως ορίσματα δίνονται η επιθυμητή ταχύτητα των κινητήρων, η στροφή του οχήματος, αν είναι επιθυμητή, και ο χρόνος για τον οποίο θα γίνει η κίνηση αυτή. Ειδικότερα, αυτό πραγματοποιείται με την έξοδο υψηλής ή χαμηλής τάσης στις GPIO κεφαλές που συνδέονται με τον Motor Controller. Ανάλογα με το τι έχει ζητηθεί από τη συνάρτηση, δηλαδή να κάνει ευθύγραμμη κίνηση ή κίνηση σε στροφή, δίνονται οι ανάλογες εντολές στους δεξιούς και αριστερούς κινητήρες. Για παράδειγμα, σε περίπτωση που (κατά την κλήση) η τιμή της μεταβλητής turn δίνεται αρνητική, τότε έχει ζητηθεί κίνηση με κατεύθυνση προς τα δεξιά, δηλαδή οι αριστεροί κινητήρες θα έχουν μεγαλύτερη ταχύτητα περιστροφής από τους δεξιούς κ.ο.κ. Η διαφορά στην ταχύτητα των δύο πλευρών ορίζεται από την τιμή της μεταβλητής turn, που σημαίνει και το πόσο απότομη ή ελαφριά στροφή θα κάνει.

```
# -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
# -- mokkasmich@gmail.com

import RPi.GPIO as GPIO
from time import sleep

class Motor():
    def __init__(self,EnaA,In1A,In2A,EnaB,In1B,In2B):
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        GPIO.cleanup()

        self.EnaA = EnaA
        self.In1A = In1A
        self.In2A = In2A
        self.EnaB = EnaB
        self.In1B = In1B
        self.In2B = In2B
        GPIO.setup(self.EnaA,GPIO.OUT)
        GPIO.setup(self.In1A,GPIO.OUT)
        GPIO.setup(self.In2A,GPIO.OUT)
        GPIO.setup(self.EnaB,GPIO.OUT)
        GPIO.setup(self.In1B,GPIO.OUT)
        GPIO.setup(self.In2B,GPIO.OUT)
        self.pwmA = GPIO.PWM(self.EnaA, 100);
        self.pwmA.start(0);
```

```
27          self.pwmB = GPIO.PWM(self.EnaB, 100);
28          self.pwmB.start(0);
29
30      def move(self,speed=0.5,turn=0,t=0):
31          speed *=100
32          turn *=100
33          leftSpeed = speed - turn
34          rightSpeed = speed + turn
35          if leftSpeed>100: leftSpeed=100
36          elif leftSpeed<-100: leftSpeed= -100
37          if rightSpeed>100: rightSpeed=100
38          elif rightSpeed<-100: rightSpeed= -100
39
40          self.pwmA.ChangeDutyCycle(abs(leftSpeed))
41          self.pwmB.ChangeDutyCycle(abs(rightSpeed))
42
43          if leftSpeed>0:
44              GPIO.output(self.In1A,GPIO.HIGH)
45              GPIO.output(self.In2A,GPIO.LOW)
46          else:
47              GPIO.output(self.In1A,GPIO.LOW)
48              GPIO.output(self.In2A,GPIO.HIGH)
49
50          if rightSpeed>0:
51              GPIO.output(self.In1B,GPIO.LOW)
52              GPIO.output(self.In2B,GPIO.HIGH)
53          else:
54              GPIO.output(self.In1B,GPIO.HIGH)
55              GPIO.output(self.In2B,GPIO.LOW)
56          if t>0:
57              sleep(t)
```

Κώδικας 6.1: Κώδικας ελέγχου Τροχών

6.3 Αναγνώριση και ακολουθία διπλής γραμμής

6.3.1 Βιβλιοθήκες εντολών που χρησιμοποιήθηκαν

Όπως αναφέρθηκε, οι βιβλιοθήκες εντολών αποτελούν χαρακτηριστικό πολλών γλωσσών προγραμματισμού και παρέχουν τη δυνατότητα με τη χρήση μίας εντολής, να μπορούν να εκτελούνται αλγόριθμοι και άλλες πράξεις που κανονικά θα χρειάζονταν πολύ κόπο και πολλές γραμμές κώδικα.

Οι βιβλιοθήκες που χρησιμοποιήθηκαν στον κώδικα που υλοποιεί την αναγνώριση και ακολουθία μιας λωρίδας (Κώδ. 6.2) είναι:

1. cv2: Η βιβλιοθήκη της OpenCV (Υποενότητα 3.2.2) που χρησιμοποιείται για εφαρμογή αλγορίθμων μηχανικής όρασης
2. numpy: Η βιβλιοθήκη NumPy που δίνει την δυνατότητα αναπαράστασης πινάκων πολλών διαστάσεων
3. time.sleep: Η εντολή sleep που διακόπτει την εκτέλεση του προγράμματος
4. math: Βιβλιοθήκη για εφαρμογή μαθηματικών πράξεων
5. MotorCode: Ο κώδικας για τον έλεγχο των κινητήρων
6. picamera: Η βιβλιοθήκη με τις εντολές για τη λειτουργία της κάμερας του raspberry pi

```
1  # -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
2  # -- mokkasmich@gmail.com
3
4  # -- Εισαγωγή Βιβλιοθηκών
5
6  import cv2
7
8  import numpy as np
9
10 import math
11
12 import MotorModule
13
14 from picamera.array import PiRGBArray
```

```
15
16 from picamera import PiCamera
17
18 import time
```

Κώδικας 6.2: Βιβλιοθήκες που χρησιμοποιήθηκαν

6.3.2 Περιγραφή αλγορίθμου αναγνώρισης λωρίδας

Algorithm 1 Αλγόριθμος Αναγνώρισης Λωρίδας Κίνησης

while True **do**

Διάβασε νέο frame από την κάμερα

Μετατροπή μεγέθους frame σε μικρότερη ανάλυση

Εφαρμογή φίλτρου θολώματος Gauss (Gaussian Blur)

Μετατροπή κωδικοποίησης εικόνας από RGB σε HSV

Δημιουργία νέας εικόνας που περιλαμβάνει μόνο τις κόκκινες αποχρώσεις

Εφαρμογή αλγορίθμου Canny για την εύρεση ακμών

Επιλογή πλαισίου ενδιαφέροντος στο κάτω μέρος της εικόνας (Region Of Interest)

Αναγνώριση τμημάτων γραμμών με τη χρήση του μετασχηματισμού του Hough

Κατανομή των τμημάτων ανάλογα με την κλίση τους στη δεξιά ή στην αριστερή γραμμή

Εύρεση της τελικής δεξιάς και αριστερής γραμμής από των συνδυασμό των κομματιών που ανήκουν σε αυτές

Υπολογισμός της γωνίας κίνησης του οχήματος με βάση την ανιχνευμένη λωρίδα

Εμφάνιση των γραμμών πάνω στο τελικό στιγμιότυπο (frame)

Οδήγηση του οχήματος

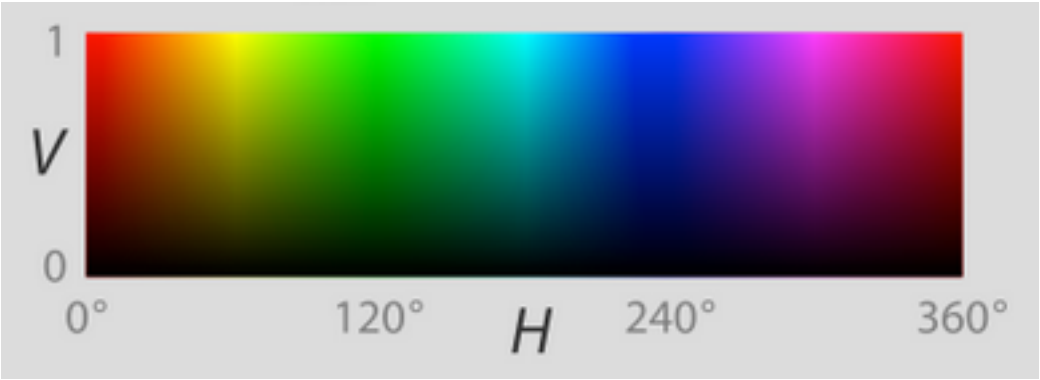
end while

6.3.3 Detect Edges

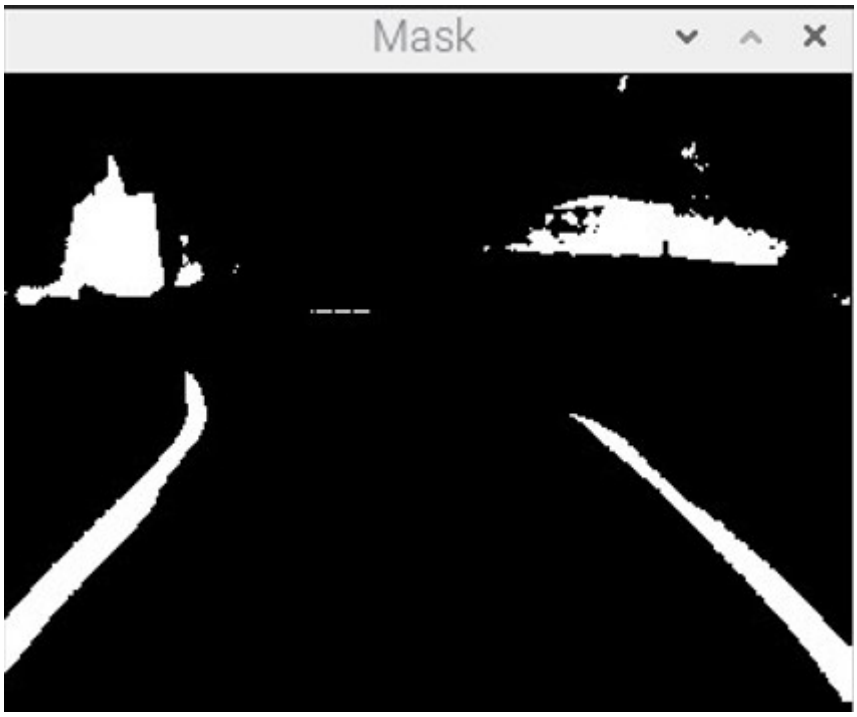
Η συνάρτηση Detect Edges (Κώδ. 6.3) είναι η πρώτη που καλείται από την κύρια συνάρτηση (main function). Είναι υπεύθυνη για ένα από τα σημαντικότερα κομμάτια του αλγορίθμου, δηλαδή την αναγνώριση των ακμών περιμετρικά των γραμμών της λωρίδας. Η συνάρτηση παίρνει ως όρισμα (δηλ. τα δεδομένα που δίνονται στην συνάρτηση κάθε φορά που καλείται) ένα στιγμιότυπο της κάμερας (ή αλλιώς frame). Αρχικά, η συνάρτηση εφαρ-

μόζει το φίλτρο θολώματος του Gauss της opencv (Υποενότητα 3.2.6), για την εξομάλυνση του θορύβου της εικόνας, χρησιμοποιώντας ένα πυρήνα (kernel) διαστάσεων 5x5. Μεγαλύτερες διαστάσεις δεν θα άλλαζαν το αποτέλεσμα, ενώ θα έκαναν πιο χρονοβόρα την διαδικασία. Έπειτα, η κωδικοποίηση της εικόνας μετατρέπεται από RGB σε HSV (Υποενότητα 3.2.4), για την ευκολότερη απομόνωση των κόκκινων αποχρώσεων, χρησιμοποιώντας την αντίστοιχη εντολή της opencv.

Έπειτα, για την απομόνωση των κόκκινων αποχρώσεων, χρησιμοποιείται χρωματική κατωφλίωση. Αναλυτικότερα, δημιουργείται μια μάσκα η οποία θα περιέχει τα εικονοστοιχεία με τιμές (Hue, Saturation και Value) που κυμαίνονται στα επιλεγμένα όρια (lower και upper red). Στη συγκεκριμένη περίπτωση, χρησιμοποιείται ο συνδυασμός δύο μασκών, καθώς το κόκκινο χρώμα βρίσκεται χωρισμένο στα δύο άκρα του χρωματικού φάσματος HSV (Σχ. 6.2). Οι τιμές που επιλέχτηκαν για τα όρια προέκυψαν από πειραματικές δοκιμές σε διάφορες καταστάσεις φωτισμού και διαπιστώθηκαν ως οι αποδοτικότερες. Η εικόνα που προκύπτει περιέχει μόνο τα pixel των γραμμών (Σχ. 6.3).

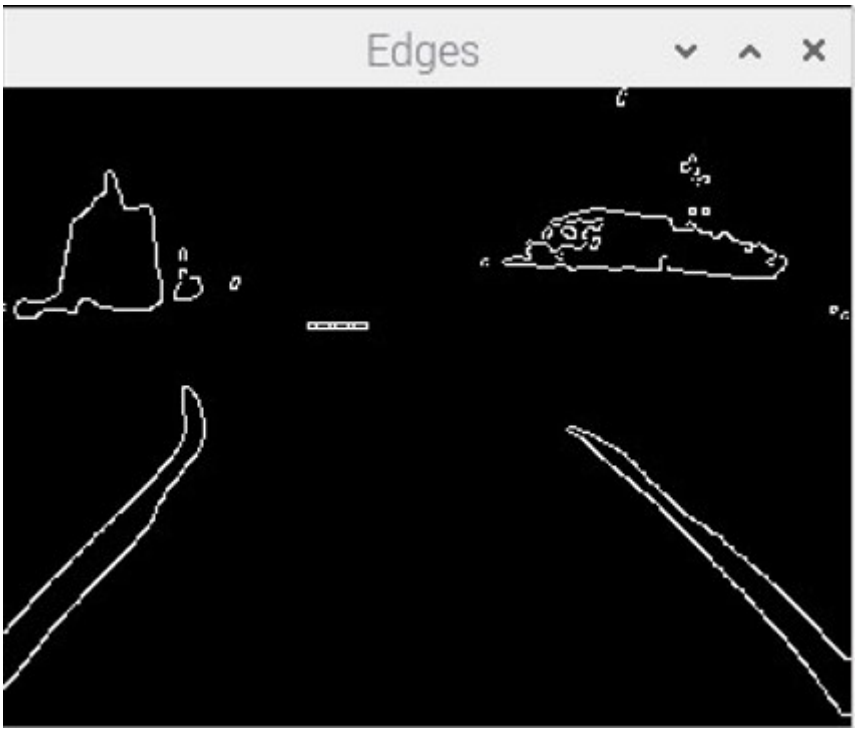


Σχήμα 6.2: Το χρωματικό φάσμα HSV [29]



Σχήμα 6.3: Η εικόνα που περιέχει μόνο τα κόκκινα pixel

Τέλος, εφαρμόζεται ο αλγόριθμος αναγνώρισης ακμών Canny (Υποενότητα 3.2.8), με τα δύο κατώφλια K1 και K2 να είναι 100 και 200, αντίστοιχα. Οι τιμές αυτές ήταν οι αποτελεσματικότερες μετά από διαδικασία πειραματικών δοκιμών. Το παραπάνω γίνεται με την αντίστοιχη εντολή της opencv. Η εικόνα που προκύπτει περιέχει μόνο τα εικονοστοιχεία ακμών, η οποία επιστρέφεται στο βασικό πρόγραμμα (Σχ. 6.4).



Σχήμα 6.4: Η εικόνα που περιέχει τις ακμές

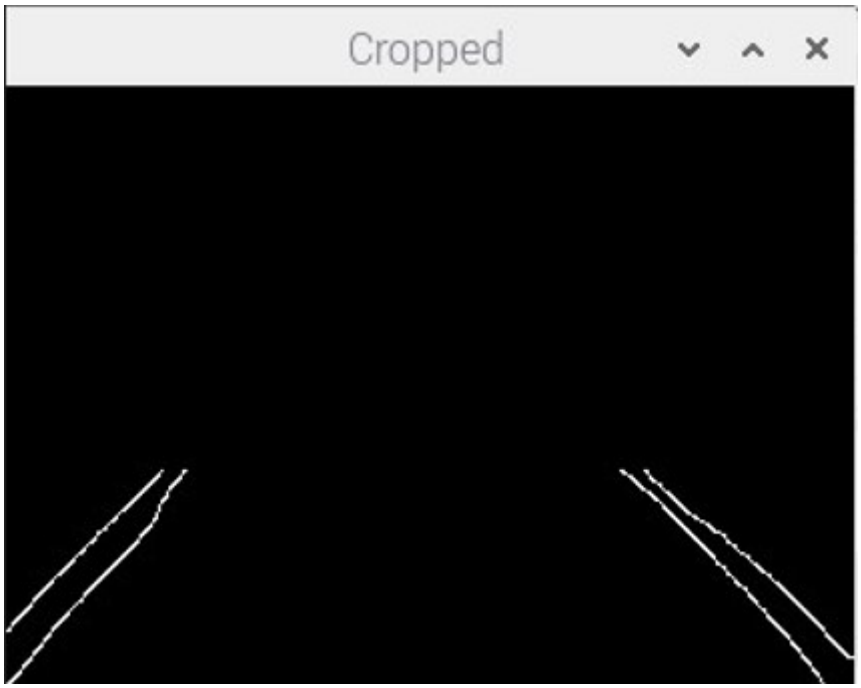
```
# -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
# -- mokkasmich@gmail.com

def image_process(frame):
    # -- Εφαρμογή φίλτρου θολώματος Gauss
    blur = cv2.GaussianBlur(frame, (5, 5), 0)

    # -- Μετατροπή της εικόνας από RGB σε HSV
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)

    # -- Δημιουργία μάσκας μόνο για τις κόκκινες αποχρώσεις
    lower_red1 = np.array([0, 50, 20])
    upper_red1 = np.array([20, 255, 255])
    mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
    lower_red2 = np.array([140, 50, 20])
    upper_red2 = np.array([180, 255, 255])
    mask2 = cv2.inRange(hsv,lower_red2,upper_red2)
    mask=mask1+mask2
    if SHOW_MASK:
        cv2.imshow("Mask",mask)
    # -- Εφαρμογή αλγορίθμου Canny για εύρεση ακμών
```

```
22      edges = cv2.Canny(mask, 100, 200)
23      return edges
```

Κώδικας 6.3: Συνάρτηση επεξεργασίας εικόνας και επιστροφής ακμών

6.3.4 Region Of Interest

Στη συνέχεια της εκτέλεσης του προγράμματος καλείται η συνάρτηση Select Region of Interest (Κώδ. 6.4), η οποία παίρνει σαν όρισμα την εικόνα με τις ακμές, όπως προκύπτει από την προηγούμενη συνάρτηση. Η επιλογή πλαισίου ενδιαφέροντος χρησιμοποιείται για να διατηρηθεί μόνο το τμήμα που είναι πιθανό να υπάρχουν γραμμές, δηλαδή το κάτω μέρος του frame, ώστε να μην πραγματοποιείται εντοπισμός κάποιων ακμών που εντοπίστηκαν στο δωμάτιο και δεν έχουν σχέση με τις ακμές. Με τη συνάρτηση αυτή επιλέγεται μόνο το κάτω μέρος της εικόνας, με 2/5 του συνολικού ύψους, η οποία είναι και η εικόνα που θα επιστραφεί για τη συνέχεια της εκτέλεσης (Σχ. 6.5).



Σχήμα 6.5: Η εικόνα των ακμών που περιέχει τα pixel που ανήκουν στο πεδίο ενδιαφέροντος

```
24  # -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
25  # -- mokkasmich@gmail.com
26
27  def select_region_of_interest(edges):
28
```

```
29      height, width = edges.shape
30      mask = np.zeros_like(edges)
31      # -- Εφαρμογή του πλαισίου από τις δύο κάτω γωνίες
32      # -- μέχρι 2/5 του συνολικού ύψους της εικόνας προς τα επάνω
33      area = np.array([[(0, height * 3/5), (width, height * 3/5), (width,
        height), (0, height),]], np.int32)
34      # -- Αντιγραφή της πληροφορίας που βρίσκεται μέσα
35      # -- στο πλαίσιο στη νέα εικόνα
36      cv2.fillPoly(mask, area, 255)
37      cropped_edges = cv2.bitwise_and(edges, mask)
38
39      return cropped_image
```

Κώδικας 6.4: Συνάρτηση επιλογής πλαισίου ενδιαφέροντος

6.3.5 Αναγνώριση γραμμών με χρήση Hough Transform

Ο αλγόριθμος, στη συνέχεια, καλεί την συνάρτηση Line Detection (Κώδ. 6.5), η οποία παίρνει σαν όρισμα την εικόνα με τις ακμές στο πλαίσιο ενδιαφέροντος. Χρησιμοποιείται μόνο μια συνάρτηση της opencv, η οποία εφαρμόζει τον Μετασχηματισμό Hough (Υποενότητα 3.2.9). Με την κλήση της, δημιουργείται μία λίστα στην οποία εισάγεται κάθε κομμάτι γραμμής που αναγνωρίζεται με βάση τα εικονοστοιχεία των ακμών από τα οποία αποτελείται. Οι παράμετροι που χρησιμοποιούνται είναι: η ακρίβεια σε pixels να είναι 1 και η ακρίβεια στις γωνιές να είναι 1 μοίρα, δηλ. π/180. Επιπλέον, ο κατώτατος αριθμός ψήφων για την γραμμή πρέπει να είναι 10 (καθώς αν ήταν μεγαλύτερος θα δυσκόλευε την αναγνώριση) και ο μικρότερος αριθμών σημείων από τον οποίο θα αποτελείται μια γραμμή πρέπει να είναι μεγαλύτερος από 30 και να απέχουν απόσταση μικρότερη από 5 pixel. Όπως φαίνεται στο Σχ. 6.6, δημιουργούνται δύο γραμμές για κάθε γραμμή, επειδή τα σημεία ακμής είναι περιμετρικά της μιας γραμμής. Αν είχε οριστεί μικρότερο μήκος (κάτω των 30 pixel) θα αναγνώριζε μικρότερες γραμμές, π.χ. κάποιον συνδυασμό μεταξύ των σημείων ακμής από τη μία και από ην άλλη πλευρά της μίας γραμμής της λωρίδας, ειδικά στο βάθος της εικόνας όπου συγκλίνουν περισσότερο. Οπότε, οι τιμές επιλέχτηκαν με πειραματική διαδικασία ως ο καλύτερος συνδυασμός.

Σχήμα 6.6: Αναγνωρισμένες γραμμές από Hough

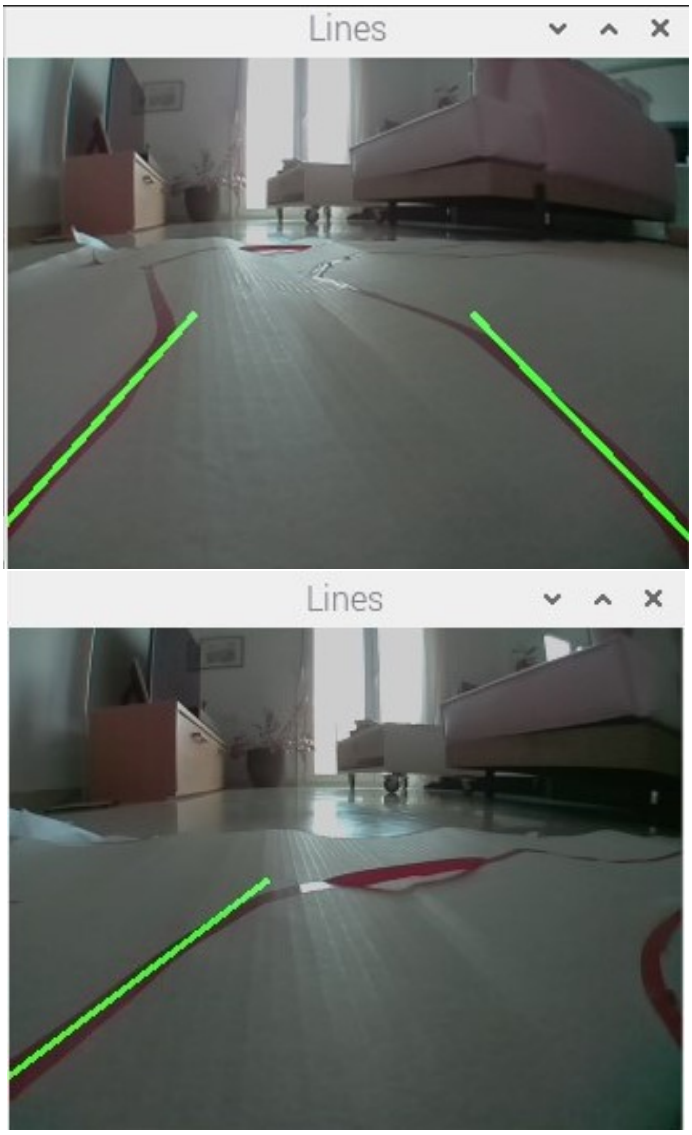
```
# -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
# -- mokkasmich@gmail.com

def line_detection(cropped_edges):
    rho = 1
    angle = np.pi / 180
    min_threshold = 10 # Κατώτατος αριθμός ψήφων για την νέα γραμμή
    minLineLength=30 # Ελάχιστος αριθμός σημείων που
                     # απαρτίζουν την γραμμή
    maxLineGap=5 # Ελάχιστος αριθμός απόστασης μεταξύ
                 # δύο διαδοχικών σημείων

    # -- Εφαρμογή Μετασχηματισμού Hough και αναγνώριση γραμμών
    # -- και αποθήκευση των συντεταγμένων
    # -- των ακραίων σημείων τους σε μια λίστα
    lines_detected = cv2.HoughLinesP(cropped_edges, rho, angle,
    min_threshold,np.array([]), minLineLength, maxLineGap)

    return lines_detected
```

Κώδικας 6.5: Συνάρτηση αναγνώρισης κομματιών από τις γραμμές της λωρίδας

6.3.6 Κατανομή με βάση την κλίση - Slope Classification

Το επόμενο βήμα του αλγορίθμου είναι η σωστή ταξινόμηση των τμημάτων των γραμμών σε αυτά που απαρτίζουν την αριστερή ή την δεξιά γραμμή της λωρίδας. Η κατανομής γίνεται με βάση την κλίση της γραμμής. Η σωστή λειτουργία βασίζεται στην προοπτική της εικόνας. Οι γραμμές που κατευθύνονται προς το βάθος συγκλίνουν νοητά στον κέντρο του ορίζοντα. Το φαινόμενο αυτό παρατηρείται εντονότερα όταν η κάμερα που χρησιμοποιείται έχει ευρυγώνια κάλυψη. Αυτός είναι και ο λόγος για τον οποίο έγινε αλλαγή της κάμερας στο όχημα, από την απλή web camera στην ειδική κάμερα του raspberry pi, με γωνία θέασης 160 μοιρών. Με την παραπάνω λογική, όποια γραμμή έχει κλίση μεγαλύτερη προς τα δεξιά σημαίνει πως ανήκει στην αριστερή γραμμή της λωρίδας, ενώ αντίθετα όποια έχει κλίση προς τα αριστερά, ανήκει στην δεξιά γραμμή της λωρίδας.

Η κατανομή αυτή επιτυγχάνεται με τη χρήση της συνάρτησης Slope Classification (Κώδ. 6.6), η οποία δέχεται σαν ορίσματα το frame και τη λίστα με τα τμήματα των γραμμών που αναγνωρίστηκαν. Αρχικά ορίζεται ένα όριο, το οποίο καθορίζει πως ένα τμήμα ανήκει στην δεξιά γραμμή, αν βρίσκεται στα δεξιά 2/3 της εικόνας, ενώ αντίστοιχα για την αριστερή, αν βρίσκεται στα αριστερά 2/3 της εικόνας. Οι περιπτώσεις κάθετης γραμμής δεν λαμβάνονται υπόψιν γιατί από τους επόμενους υπολογισμούς προκύπτει άπειρο. Οι υπολογισμοί αυτοί αφορούν την διαδικασία εύρεσης του πολυωνύμου πρώτου βαθμού που αντιστοιχεί στη γραμμή. Στην περίπτωση της κάθετης γραμμής, απειρίζεται η τιμή b η οποία είναι το σημείο τομής του πολυωνύμου με τον άξονα y (Εξίσωση πολυωνύμου πρώτου βαθμού: $y = ax + b$). Η εύρεση του πολυωνύμου γίνεται με την έτοιμη συνάρτηση polyfit, η οποία μας επιστρέφει τις τιμές a και b, όπου a η κλίση της γραμμής. Έτσι, μια γραμμή ορίζεται ότι ανήκει στην δεξιά γραμμή αν η τιμή της κλίσης είναι θετική, αλλά και τα δύο άκρα της γραμμής είναι δεξιά από το όριο που ορίσαμε στην αρχή(δεξιά 2/3 της εικόνας). Ενώ, αντίστοιχα ανήκει στην αριστερή αν η κλίση είναι αρνητική και τα δύο άκρα της είναι στο αριστερό όριο. Τέλος, υπολογίζεται ο μέσος όρος των γραμμών της αριστερής και της δεξιάς λίστας, με βάση τις τιμές a και b και καθορίζονται τα άκρα της τελικής αριστερής και δεξιάς γραμμής, τα οποία ξεκινάνε από το κάτω μέρος και φτάνουν μέχρι το μέσο του ύψους του frame (Σχ. 6.7).



Σχήμα 6.7: Περιπτώσεις με μία και δύο αναγνωρισμένες γραμμές

```
58 # -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
59 # -- mokkasmich@gmail.com
60
61 def slope_classification(frame, line_segments):
62
63     # -- Έλεγχος αν η λίστα που έχει δοθεί είναι άδεια
64     # -- σημαίνει πως δεν έχουν αναγνωριστεί γραμμές
65     lane_lines = []
66     if line_segments is None:
67         return lane_lines
68
69     height, width, _ = frame.shape
```

```
70
71      # -- Δημιουργία δύο λιστών που περιέχουν τις γραμμές που
    ταξινομούνται
72      # -- στη δεξιά και αριστερή γραμμή της λωρίδας
73      left_fit = []
74      right_fit = []
75
76      boundary = 1/3
77      # -- Καθορισμός ορίου: η δεξιά γραμμή πρέπει να βρίσκεται
78      # -- στα δεξιά 2/3 της εικόνας
79      # -- ενώ η αριστερή στα αριστερά 2/3
80      left_region_boundary = width * (1 - boundary)
81      right_region_boundary = width* boundary
82
83      # -- Επανάληψη για κάθε γραμμή της λίστας
84      for line_segment in line_segments:
85          for x1, y1, x2, y2 in line_segment:
86
87              # -- Αποφυγή κάθετης γραμμής γιατί θα προκύψει απειρο
88              if x1 == x2:
89                  continue
90              # -- Υπολογισμός πολυωνύμου πρώτου βαθμού
91              # -- της αναγνωρισμένης γραμμής
92              fit = np.polyfit((x1, x2), (y1, y2), 1)
93
94              # -- Η κλίση της γραμμής
95              slope = fit[0]
96              # -- Το σημείο που τέμνει τον κατακόρυφο άξονα
97              # -- η προέκταση της γραμμής
98              intercept = fit[1]
99
100             # -- Αν η κλίση αρνητική, τότε ανήκει
101             # -- στην αριστερή γραμμή της λωρίδας
102             if slope < 0:
103                 if x1 < left_region_boundary and x2 <
    left_region_boundary:
104                     left_fit.append((slope, intercept))
105             # -- Αν η κλίση θετική, τότε ανήκει
106             # -- στην δεξιά γραμμή της λωρίδας
```

```
107             else:
108                 if x1 > right_region_boundary and x2 >
    right_region_boundary:
109                     right_fit.append((slope, intercept))
110
111     # -- Υπολογισμός της μέσης γραμμής από όλες τις γραμμές
112     # -- που ανήκουν στην αριστερή γραμμή της λωρίδας
113     left_fit_average = np.average(left_fit, axis=0)
114     if len(left_fit) > 0:
115         lane_lines.append(make_points(frame, left_fit_average))
116
117     # -- Υπολογισμός της μέσης γραμμής από όλες τις γραμμές
118     # -- που ανήκουν στην δεξιά γραμμή της λωρίδας
119     right_fit_average = np.average(right_fit, axis=0)
120     if len(right_fit) > 0:
121         lane_lines.append(make_points(frame, right_fit_average))
122
123     return lane_lines
```

Κώδικας 6.6: Κατανομή των ανιχνευμένων γραμμών σε δεξιά και αριστερή γραμμή της λωρίδας

6.3.7 Εμφάνιση των Γραμμών - Display lines

Στη συνέχεια, γίνεται αναπαράσταση των δύο ανιχνευμένων γραμμών της λωρίδας πάνω στο frame, με χρήση της συνάρτησης Display Lines (Κώδ. 6.7). Η συνάρτηση λαμβάνει ως ορίσματα το frame και τις δύο ανιχνευμένες γραμμές, καθώς επίσης, και τις σχετικές επιλογές για το χρώμα και το πάχος των γραμμών που θα τις αναπαραστήσουν. Η αναπαράσταση των γραμμών είναι σημαντική για την παρατήρηση και τον έλεγχο της σωστής λειτουργίας του οχήματος όπως και για την επίλυση των προβλημάτων που προέκυπταν κατά την ανάπτυξη του αλγορίθμου. Στη συνέχεια, προστέθηκε ένας έλεγχος ο οποίος ήταν μεγάλης σημασίας για την σωστή λειτουργία του αλγορίθμου. Ο έλεγχος αυτός, αποτρέπει την αναγνώριση δύο γραμμών λωρίδας, ενώ στην πραγματικότητα υπάρχει μόνο μία στο frame. Αυτό συμβαίνει όταν η γραμμή της λωρίδας έχει μεγάλο πάχος, βρίσκεται στο μέσο του frame και εκτείνεται προς το βάθος. Έτσι, οι δύο πλευρές της γραμμής λόγω της προοπτικής του βάθους έχουν διαφορετική κλίση και αναγνωρίζονται ως δύο διαφορετικές (Σχ. 6.8). Αυτό έχει σαν απο-

τέλεσμα το όχημα να προσπαθεί να ακολουθήσει αυτές τις γραμμές και να προκύψει λάθος στη συνέχεια. Το παραπάνω μπορεί να αποφευχθεί κάνοντας κάποιους ελέγχους για την θέση των γραμμών και την μεταξύ τους απόσταση, όπως για παράδειγμα αν η απόσταση των πιο απομακρυσμένων σημείων των γραμμών είναι μικρότερη των 50 pixels, τότε κρατάμε την μία από τις δύο γραμμές. Αν το μεγαλύτερο μέρος της μίας γραμμής ανήκει ολόκληρο στο αριστερό μισό frame τότε διατηρείται μόνο αυτό και θα αποτελεί την αριστερή γραμμή. Το ίδιο ισχύει, αντίστοιχα, και για την δεξιά γραμμή. Στη συνέχεια, με τη χρήση της εντολής line της opencv, μπορούμε να αναπαραστήσουμε τις γραμμές ή την γραμμή της λωρίδας επάνω στον καμβά του frame. Τέλος, επιστρέφεται το frame με τις γραμμές αποτυπωμένες επάνω του (Σχ. 6.7).

Σχήμα 6.8: Πρόβλημα αναγνώρισης μίας ως διπλής γραμμής

```
125 # -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
126 # -- mokkasmich@gmail.com
127
128 def display_lines(frame, lines, line_color=(0, 255, 0), line_width=2):
129     line_image = np.zeros_like(frame)
130
131     # -- Αποφυγή αναγνώρισης μιας γραμμής λόγω τους πάχους σαν δύο,
132     # -- ελέγχοντας την απόσταση και τη θέση των γραμμών,
133     # -- και διατήρηση μόνο της μίας από τις δύο,
134     # -- ανάλογα με την τοποθεσία της
```

```
135     # -- Τέλος ακολουθεί ο σχεδιασμός των γραμμών που επιτράπηκαν
136     if len(lines)==2:
137         [[left_x1,left_y1,left_x2,left_y2]] = lines[0]
138         [[right_x1,right_y1,right_x2,right_y2]] = lines[1]
139         if abs(left_x2-right_x2)<50:
140             if left_x1<220:
141                 lines.pop(1)
142             else:
143                 lines.pop(0)
144             [[x1,y1,x2,y2]] = lines[0]
145             cv2.line(line_image, (x1,y1), (x2, y2), line_color,
    line_width)
146         else:
147             cv2.line(line_image, (left_x1, left_y1), (left_x2, left_y2),
    line_color, line_width)
148             cv2.line(line_image, (right_x1, right_y1), (right_x2,
    right_y2), line_color, line_width)
149     # -- Περίπτωση Αναγνώρισης μίας μόνο γραμμής
150     elif len(lines)==1:
151         [[x1,y1,x2,y2]] = lines[0]
152         cv2.line(line_image, (x1,y1), (x2, y2), line_color, line_width)
153 
154     line_image = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
155     return line_image
```

Κώδικας 6.7: Έλεγχος λανθασμένης ανίχνευσης δύο γραμμών και σχεδίαση τους επάνω στο frame

6.3.8 Steering angle

Το τελευταίο βήμα του αλγορίθμου ανίχνευσης λωρίδας είναι ο υπολογισμός της γωνίας της πορείας του οχήματος, βασιζόμενος στις αναγνωρισμένες γραμμές της λωρίδας. Αυτό επιτυγχάνεται με την κλήση της συνάρτησης Steer (Κώδ. 6.8), η οποία παίρνει σαν ορίσματα το frame και τις αναγνωρισμένες γραμμές και στη συνέχεια, επιστρέφει την κλίση της πορείας του οχήματος και το τελικό frame του αλγορίθμου. Η συνάρτηση καλεί δύο υποσυναρτήσεις, ώστε να διεκπεραιώσουν τις παραπάνω διαδικασίες.

Αρχικά, καλείται η Compute Steering Angle, η οποία υπολογίζει την κλίση της γραμ-

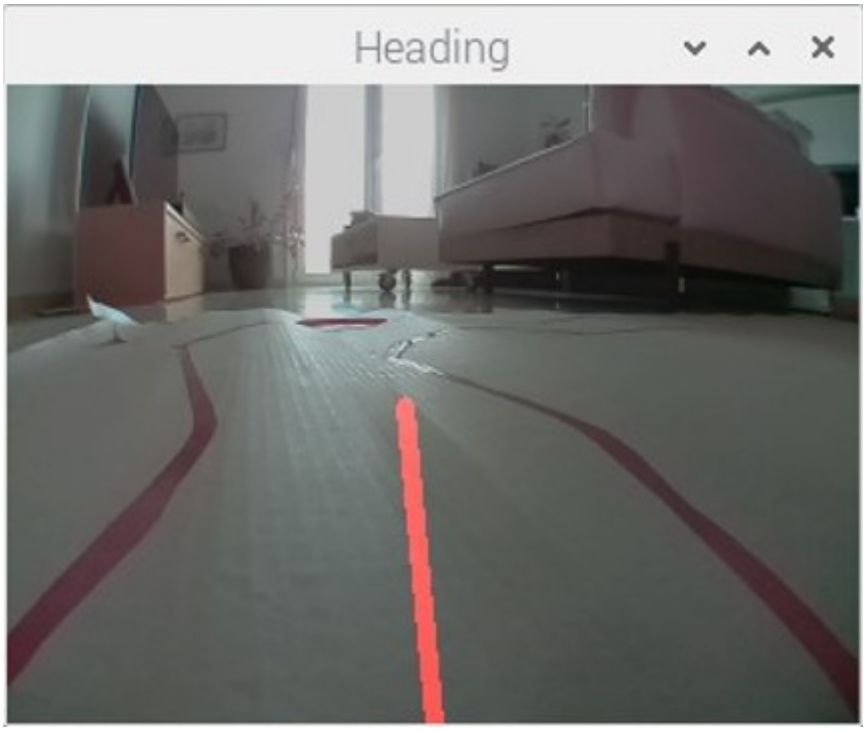
μής πορείας, με βάση τις γραμμές που αναγνωρίστηκαν στο προηγούμενο στάδιο. Υπάρχουν τρεις περιπτώσεις αναγνωρισμένων γραμμών.

1. Να μην έχουν ανιχνευθεί καθόλου γραμμές, οπότε η συνάρτηση επιστρέφει την προκαθορισμένη τιμή και δεν υπάρχει γραμμή πορείας για το όχημα.

2. Να υπάρχει μόνο μία αναγνωρισμένη γραμμή, οπότε το όχημα θα ακολουθήσει πορεία παράλληλη με την αναγνωρισμένη γραμμή και θα επιστρέψει την κλίση αυτής της γραμμής.

3. Να υπάρχουν δύο γραμμές, όπου η γραμμή πορείας θα έχει κλίση ίση με τη μέση κλίση των δύο γραμμών. Η κλίση που επιστρέφεται είναι υπολογισμένη σε μοίρες και η τιμή 90 συμβολίζει την κατακόρυφη γραμμή πορείας.

Στη συνέχεια, καλείται η συνάρτηση Display Heading Line που παίρνει την κλίση της γραμμής πορείας του οχήματος την οποία σχεδιάζει πάνω στο frame και την επιστρέφει. Η γραμμή πορείας που σχεδιάζεται σε κάθε frame, ξεκινάει πάντα από ένα σταθερό σημείο στο μέσον του κάτω μέρους του frame και εκτείνεται μέχρι το μισό του ύψους του frame. Οπότε, το δεύτερο σημείο της γραμμής υπολογίζεται από την ήδη γνωστή κλίση που πρέπει να έχει η γραμμή πορείας. Τέλος, με την συνάρτηση της opencv σχηματίζει την γραμμή στο frame και το επιστρέφει (Σχ. 6.9).



Σχήμα 6.9: Η γραμμή πορείας του οχήματος

```
156 # -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
157 # -- mokkasmich@gmail.com
158
159 def steer(frame, lane_lines):
160
161     # -- Περίπτωση καμίας αναγνωρισμένης γραμμής
162     if len(lane_lines) == 0:
163         return frame
164
165     new_steering_angle = compute_steering_angle(frame, lane_lines)
166     curr_heading_image = display_heading_line(frame, new_steering_angle)
167
168     return curr_heading_image, new_steering_angle
169
170 def compute_steering_angle(frame, lane_lines):
171     # -- Περίπτωση καμίας γραμμής
172     if len(lane_lines) == 0:
173         return -90
174     height, width, _ = frame.shape
175
176     # -- Περίπτωση αναγνώρισης μίας γραμμής
```

```
177     if len(lane_lines) == 1:
178         x1, _, x2, _ = lane_lines[0][0]
179         x_offset = x2 - x1
180 
181     # -- Περίπτωση αναγνώρισης δύο γραμμών
182     else:
183         _, _, left_x2, _ = lane_lines[0][0]
184         _, _, right_x2, _ = lane_lines[1][0]
185         camera_offset = 0.02 # -- Κανονικοποίηση για κεντράρισμα
186         mid = int(width / 2 * (1 + camera_offset))
187         x_offset = (left_x2 + right_x2) / 2 - mid
188 
189     # -- Υπολογισμός γραμμής πορείας του οχήματος στο μέσο
190     # -- των δύο γραμμών της λωρίδας
191 
192     y_offset = int(height / 2)
193 
194     angle_to_mid_radian = math.atan(x_offset / y_offset)
195 
196     # -- Μετατροπή από ακτίνια σε μοίρες
197     angle_to_mid_deg = int(angle_to_mid_radian * 180.0 / math.pi)
198     steering_angle = angle_to_mid_deg + 90
199 
200     return steering_angle
201 
202 
203 def display_heading_line(frame, steering_angle, line_color=(0, 0, 255),
        line_width=5 ):
204     heading_image = np.zeros_like(frame)
205     height, width, _ = frame.shape
206 
207     steering_angle_radian = steering_angle / 180.0 * math.pi
208     x1 = int(width / 2)
209     y1 = height
210     x2 = int(x1 - height / 2 / math.tan(steering_angle_radian))
211     y2 = int(height / 2)
212 
213     cv2.line(heading_image, (x1, y1), (x2, y2), line_color, line_width)
214     heading_image = cv2.addWeighted(frame, 0.8, heading_image, 1, 1)
```

```
215
216     return heading_image
```

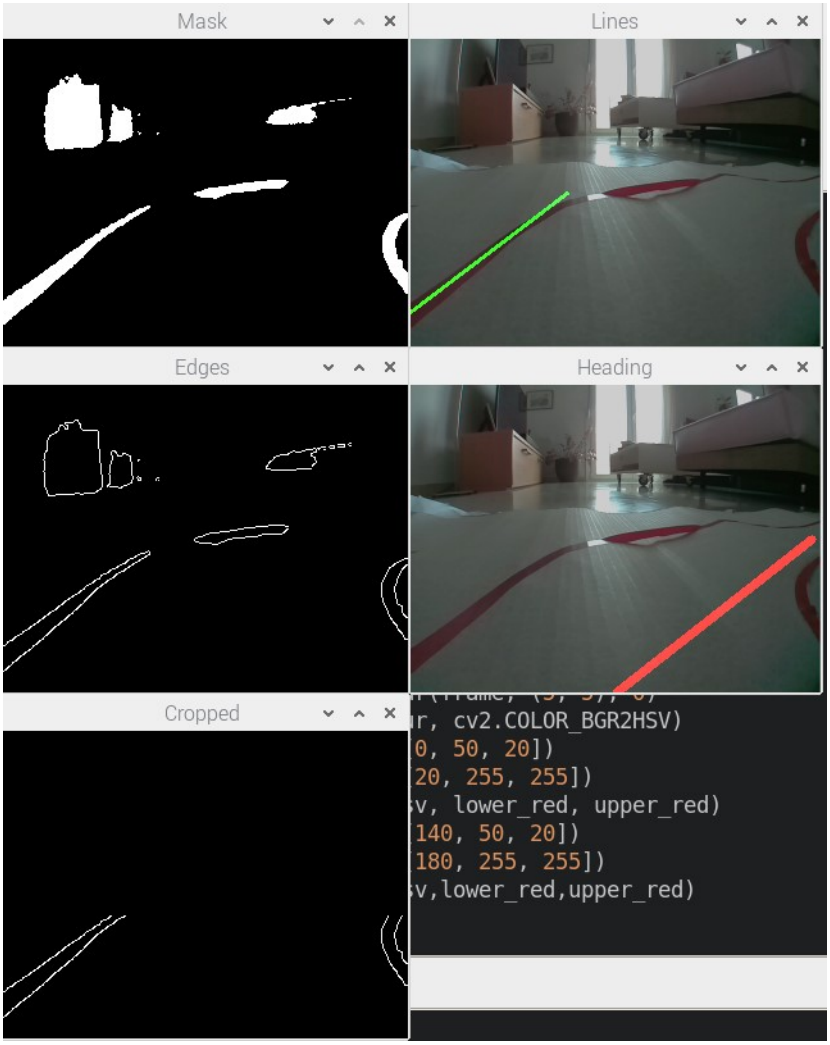
Κώδικας 6.8: Υπολογισμός της γωνίας της πορείας του οχήματος μέσα στην λωρίδα κυκλοφορίας και σχεδιασμός της γραμμής στο frame

6.3.9 Κύρια συνάρτηση - Main Function

Η κύρια συνάρτηση ή αλλιώς main function (Κώδ. 6.9) είναι η συνάρτηση η οποία καλείται πρώτη με την εκτέλεση του προγράμματος και στη συνέχεια η ίδια καλεί με την σειρά που αναφέρθηκαν παραπάνω τις υπόλοιπες συναρτήσεις. Η κλήση όλων των παραπάνω συναρτήσεων γίνεται επαναληπτικά για κάθε frame της κάμερας μέχρι να ζητηθεί λήξη του προγράμματος με το πάτημα του πλήκτρου «q».

Στην αρχή, γίνεται η αρχικοποίηση των ρυθμίσεων της κάμερας. Ο αλγόριθμος στα πρώτα στάδια ανάπτυξής του χρησιμοποιούσε ανάλυση 640x480 και υπήρχε μια καθυστέρηση μεταξύ των frames. Σε περίπτωση απότομης στροφής, έπρεπε το όχημα να έχει μικρή ταχύτητα ώστε να την ακολουθήσει frame προς frame. Αν το όχημα είχε μεγαλύτερη ταχύτητα, τότε από το ένα frame μέχρι το επόμενο θα υπήρχε ένα κενό και το όχημα θα είχε συνεχίσει ευθεία χάνοντας την στροφή, καθώς ακόμα θα επεξεργαζόταν το προηγούμενο frame. Για το λόγο αυτό, χρησιμοποιήθηκε ανάλυση 320x240, δηλαδή υπάρχουν τέσσερις φορές λιγότερα pixel. Με τη διαδικασία αυτή μειώνεται σημαντικά ο χρόνος επεξεργασίας του κάθε frame, και δίνεται η δυνατότητα για μεγαλύτερη ταχύτητα στο όχημα, διατηρώντας ωστόσο τη σταθερότητα του στις απότομες στροφές.

Το επαναλαμβανόμενο loop ξεκινάει από τη γραμμή 211, και ο κώδικας που βρίσκεται μέσα σε αυτό θα επαναλαμβάνεται για κάθε frame. Στην γραμμή κώδικα 237, γίνεται ειδικός έλεγχος στην περίπτωση μιας γραμμής (πράγμα που συμβαίνει, κυρίως, όταν ακολουθεί μια απότομης κλίσης στροφή) για να αποφευχθεί η στροφή του οχήματος νωρίτερα, ελέγχοντας αν το κατώτατο άκρο της γραμμής έχει τιμές εκτός του frame, που σημαίνει ότι το όχημα πλησιάσει την στροφή αρκετά ώστε να ξεκινήσει να στρίβει (Σχ. 6.10).



Σχήμα 6.10: Περίπτωση απότομης στροφής με μία ανιχνευμένη γραμμή

Μετά την επιστροφή της κλίσης της γραμμής πορείας του οχήματος από την συνάρτηση steer, γίνεται ο έλεγχος για την κίνηση του οχήματος. Για τον έλεγχο των τροχών κίνησης χρησιμοποιείται, όπως αναφέρθηκε παραπάνω, ο κώδικας 6.1 και οι τροχοί έχουν αρχικοποιηθεί στην πρώτη γραμμή της main. Έπειτα, για τη κάθε περίπτωση στροφής, υπάρχουν δύο περιπτώσεις. Η πρώτη, με πιο απότομη κλίση, οπότε θα στρίψει το όχημα με μεγαλύτερη ένταση και η δεύτερη, με μικρότερη κλίση, οπότε θα γίνει πιο ομαλή στροφή του οχήματος. Τέλος, εάν δεν υπάρχει ανίχνευση γραμμών για κάποιο χρονικό διάστημα, συμπεραίνεται ότι έχει υπάρξει κάποια παρεκτροπή από την πορεία της λωρίδας, οπότε το όχημα θα κάνει όπισθεν ώστε να ξαναμπεί στην λωρίδα.

```
218 # -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
219 # -- mokkasmich@gmail.com
220
```

```
motor = MotorModule.Motor(4,3,2,22,17,27)
try:
#-- Αρχικοποίηση ρυθμίσεων κάμερας
    camera = PiCamera()
    camera.resolution = (320, 240)
    camera.brightness = 55
    rawCapture = PiRGBArray(camera, size=(320, 240))
    time.sleep(0.5)
    count=0

#-- Το κύριο loop το οποίο παίρνει ένα ένα τα frames για την επεξεργασία
    while True:
        #-- Σύλληψη του frame από την κάμερα σε κωδικοποίηση RGB
        camera.capture(rawCapture, format="bgr", use_video_port=True)
        frame = rawCapture.array
        timer = cv2.getTickCount() #Μετρητής-- για υπολογισμό frames ανά
    δευτερόλεπτο

        # -- Κλήση της συνάρτησης που επεξεργάζεται το frame
        # -- και επιστρέφει μια εικόνα με μόνο τις ακμές
        edges = image_process(frame)

        # -- Ορισμός πλαισίου που θα γίνεται η ανίχνεσυη λωρίδας
        cropped_edges = select_region_of_interest(edges)

        # -- Ανίχνευση για γραμμές
        line_segments = line_detection(cropped_edges)

        # -- Κατανομή των γραμμών σε δεξιά και αριστερή
        lane_lines = slope_classification(frame, line_segments)

        # -- Δημιουργία frame με τις ανιχνευμένες γραμμές
        lane_lines_image = display_lines(frame, lane_lines)

        # -- Περίπτωση που ανιχνεύεται μία γραμμή να μη στρίβει
        # -- πολύ νωρίς σε περίπτωση απότομης κλίσης
        if len(lane_lines)==1:
            for line in lane_lines:
                [[x1, y1, x2, y2]]=line
```

```
259                 if x1<-20 or x1>320:
260                     print(”HIGH”)
261                     TURN=0
262                 else:
263                     TURN=1
264 
265         # -- Περίπτωση που ανιχνεύονται γραμμές στο frame
266         if len(lane_lines)>0:
267 
268             # -- Υπολογιμός πορείας οχήματος με βάση τη κλίση της λωρίδας
269             combo_image, new_steering_angle = steer(frame, lane_lines)
270 
271             # -- Εκτύπωση στο frame την κλίση
272             cv2.putText(combo_image, str(new_steering_angle), (50,50),
    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 1, cv2.LINE_AA)
273 
274             # -- Εντολές για την κίνηση των τροχών αναλόγως την περίπτωση
275             if MOTORS_ON:
276                 # -- Στροφή προς τα δεξιά
277                 if new_steering_angle > 160 and TURN:
278                     motor.move(0.1, -0.4,  0)#0.1 , -0.4
279                     count=0
280                     print('Steep Right')
281                 elif new_steering_angle > 110and TURN:
282                     motor.move(0.1, -0.25,  0)
283                     count=0
284                     print('Right')
285                 # -- Στροφή προς τα αριστερά
286                 elif new_steering_angle < 20 and TURN:
287                     count=0
288                     motor.move(0.1, 0.4, 0) #0.1 , 0,4
289                     print('Steep Left')
290                 elif new_steering_angle < 70 and TURN:
291                     count=0
292                     motor.move(0.1, 0.25, 0)
293                     print('Left')
294                 else:
295                     count=0
296                     motor.move(0.15, 0, 0)
```

```
297
298         else:
299         # -- Σε περίπτωση μη αναγνώρισης καμίας γραμμής για κάποιο χρόνο,
300         # -- οπισθέν για να ξαναμπεί στη λεωρίδα εφόσον βγήκε
301             count=count+1
302             if MOTORS_ON:
303                 if count==15:
304                     motor.move(-0.3,0,0.2)
305                     count=0
306             motor.stop(0)
307             combo_image = steer(frame, lane_lines)
308
309         if _SHOW_IMAGE:
310         # -- Εμφάνιση της εικόνας του frame με τις ανιχνευμένες γραμμές,
311         # -- την κλίση της πορείας και τα fps
312             fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer);
313             cv2.putText(lane_lines_image, "FPS : " + str(int(fps)), (100,
    50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);
314
315             resized = cv2.resize(lane_lines_image,(640,480))
316             cv2.imshow("Road with Lane line", resized)
317
318 #Προετοιμασία για να πάρει το νεότερο frame στο επόμενο loop
319
320         rawCapture.truncate(0)
321 #Έλεγχος λήξης προγράμματος άμα πατηθεί το q
322         if cv2.waitKey(1) & 0xFF == ord('q'):
323             motor.stop(0)
324             break
325 finally:
326     cv2.destroyAllWindows()
```

Κώδικας 6.9: Η main συνάρτηση του προγράμματος

Κεφάλαιο 7

Συμπεράσματα

7.1 Σύνοψη και συμπεράσματα

Η σύγχρονη εποχή χαρακτηρίζεται από αλματώδη και συνεχή τεχνολογική ανάπτυξη. Βρισκόμαστε στην αρχή μιας νέας εποχής, αυτή της αυτόνομης οδήγησης. Η παρούσα διπλωματική αφορά την κατασκευή ενός αυτόνομου ρομποτικού οχήματος, ενισχυμένου με ένα ρομποτικό βραχίωνα, και την αυτόνομη οδήγησή του με την υλοποίηση ενός αλγόριθμου ανίχνευσης λωρίδας πραγματικού χρόνου. Στο πρώτο μέρος αναφέρθηκαν και αναλύθηκαν διάφοροι αλγόριθμοι μηχανικής όρασης που έχουν υλοποιηθεί για την ανίχνευση μιας λωρίδας δρόμου. Κάποιοι από αυτούς επιχειρούσαν την καλύτερη και αποδοτικότερη ανίχνευση λωρίδας, συνδυάζοντας γνωστές τεχνικές με εφαρμογές μηχανικής μάθησης, ενώ άλλες παρουσίασαν πρωτοποριακές προσεγγίσεις για την λύση του προβλήματος. Ωστόσο, η πλειοψηφία δοκιμάστηκε σε βιντεοσκοπημένες διαδρομές. Σε αντίθεση με αυτές, ο σκοπός της εργασίας ήταν η υλοποίηση ενός αλγόριθμου πραγματικού χρόνου που δοκιμάστηκε σε πραγματικό όχημα.

Στην παρούσα εργασία παρουσιάστηκε η κατασκευή ενός οχήματος με τέσσερις τροχούς και ένα μεγάλου μεγέθους ρομποτικό βραχίωνα επάνω στη βάση του. Έπειτα, με βάση τεχνικές που έχουν χρησιμοποιήσει και αρκετές από τις αναφερόμενες έρευνες υλοποιήθηκε και αναλύθηκε ένας αλγόριθμος για την επιτυχή ανίχνευση λωρίδας σε πραγματικό χρόνο. Πραγματοποιήθηκε επεξήγηση της λειτουργίας του κώδικα, ο οποίος ήταν υπεύθυνος για την πρακτική υλοποίηση του αλγόριθμου. Η δοκιμή έγινε πάνω στο όχημα που διαθέτει δικό του σύστημα επεξεργασίας, κάνοντάς το πλήρως αυτόνομο.

7.2 Μελλοντικές επεκτάσεις

Στα πλαίσια της διπλωματικής εργασίας έγινε μία προσπάθεια να καταδειχθεί η σημασία της αυτόνομης οδήγησης στο σύγχρονο κόσμο. Η προώθηση νέων τεχνολογιών θα βελτιώσει και θα διευκολύνει την καθημερινότητα των ανθρώπων. Με την ολοκλήρωση της εργασίας αυτής αναπτύχθηκε ένας αλγόριθμος αναγνώρισης λωρίδας που δοκιμάστηκε σε αυτοδημιούργητο όχημα και ήταν επιτυχής σε συνθήκες πραγματικού χρόνου. Ωστόσο, προτείνεται και η διεξαγωγή περαιτέρω έρευνας μελλοντικά χρησιμοποιώντας ιδέες και τεχνικές που καταγράφηκαν σε άλλες έρευνες.

Μία πρόταση αναβάθμισης του οχήματος θα ήταν η χρήση ισχυρότερου επεξεργαστή για την δοκιμή διαφόρων πρωτοποριακών τεχνικών, όπως αυτών που αναφέρθηκαν, που χρήζουν μεγαλύτερης επεξεργαστικής ισχύος για την λειτουργία τους σε πραγματικό χρόνο. Επιπροσθέτως, με καλύτερους πόρους από αυτούς του παρόντος εγχειρήματος, θα ήταν δυνατή η χρήση και άλλων αισθητήρων, με σημαντικότερη το LiDAR. Με αυτόν τον τρόπο, θα γινόταν δυνατή η τρισδιάστατη απεικόνιση του περιβάλλοντα χώρου. Το όχημα θα αντιλαμβάνεται, πλέον, όλα τα υπόλοιπα οχήματα που, πιθανόν, κινούνται γύρω του, χωρίς την ανάγκη ύπαρξης καλής ποιότητας γραμμής κυκλοφορίας. Θα δίνεται η δυνατότητα αναγνώρισης πεζοδρομίου και κώνων σήμανσης ως όρια του δρόμου, ενώ, τέλος, θα γίνεται εύκολα η αναγνώριση των πινακίδων σήμανσης οδικής κυκλοφορίας. Επίσης, θα αποφευχθούν οι ανακρίβειες στους υπολογισμούς λόγω καιρικών συνθηκών και φωτισμού αλλά και των υπόλοιπων περιορισμών που έχει η χρήση μίας απλής κάμερας.

Μια ακόμα επέκταση της εργασίας θα ήταν η χρήση του ρομποτικού βραχίωνα για την επιτέλεση διαφόρων αυτόνομων εργασιών, όπως ο χρωματισμός των γραμμών της λωρίδας, η απομάκρυνση φυσικών εμποδίων από το οδόστρωμα και, γενικότερα, η βοήθεια σε έργα οδικών επισκευών και οδοποιίας. Τέλος, ο συνδυασμός του οχήματος με ένα σύστημα πλοήγησης GPS και η χρήση ενός μεγαλύτερου οχήματος θα καθιστούσε δυνατή την παράδοση δεμάτων σε πολλαπλούς παραλήπτες. Ο τελικός στόχος θα ήταν η δοκιμή των τεχνικών που διερευνήθηκαν σε οχήματα κανονικού μεγέθους σε φυσικούς δρόμους κυκλοφορίας.

Βιβλιογραφία

[1] Satyam Kalan, Sanket Chauhan, Rafael F. Coellho, Marcelo A. Orvieto, Ignacio R. Camacho, Kenneth J. Palmer, and Vipul R. PAtel. History of robotic surgery. *Springer*, July 2010.

[2] Standford. `https://cs.stanford.edu/groups/manips/teaching/cs225a/projects.html`. Ημερομηνία πρόσβασης: 12-05-2021.

[3] Nix color sensor. `https://www.nixsensor.com/blog/what-is-rgb-color/`. Ημερομηνία πρόσβασης: 22-07-2021.

[4] Keith carter. `https://www.keithbcarter.com/think-differently-blog/how-a-machine-sees`. Ημερομηνία πρόσβασης: 22-07-2021.

[5] Dibya Jyoti Bora, Anil Kumar Gupta, and Fayaz Ahmad Khan. Comparing the performance of l*a*b and hsv color spaces with respect to color image segmentation. *International Journal of Emerging Technology and Advanced Engineering*, 5(2):192–203, February 2015.

[6] Aisha Ajmal, Christopher Hollitt, Marcus Frean, and Harith Al-Sahaf. A comparison of rgb and hsv colour spaces for visual attention models. Technical report, School of Engineering and Computer Science Victoria University of Wellington Wellington, NewZealand, 2018.

[7] Paco. `https://paco-cpu.github.io/paco-cpu/docs/eval-gauss/`. Ημερομηνία πρόσβασης: 28-07-2021.

[8] Data carpentry. `https://datacarpentry.org/image-processing/`. Ημερομηνία πρόσβασης: 28-07-2021.

[9] Pathmanabhan A and Dinesh Sathyamoorthy. The effect of gaussian blurring on the extraction of peaks and pits from digital elevation models. *Discrete Dynamics in Nature and Society*, 2007, 01 2007.

[10] Tom robotics. `https://tomrobotics.altervista.org`. Ημερομηνία πρόσβασης: 28-07-2021.

[11] Justin liang. `https://justin-liang.com/tutorials/canny/#suppression`. Ημερομηνία πρόσβασης: 30-07-2021.

[12] Towards data science. `https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549`. Ημερομηνία πρόσβασης: 2-08-2021.

[13] Ε. Μαραγκουδάκης. Εύρεση Τροχιών Σε Ανιχνευτές Micromegas Με Τη Χρήση hough transform. Πτυχιακή εργασία, Τμήμα Φυσικής, Τομέας Πυρηνικής Φυσικής και Φυσικής Στοιχειωδών Σωματιδίων, Φεβρ. 2016.

[14] Data hacker. `http://datahacker.rs/opencv-line-detection-using-hough-transform/`. Ημερομηνία πρόσβασης: 2-08-2021.

[15] Medium: Road lane lines detection using advanced computer vision techniques. `https://medium.com/@mithi/advanced-lane-finding-using-computer-vision-techniques-7f3230b6c6f2`. Ημερομηνία πρόσβασης: 3-08-2021.

[16] Medium: Opencv perspective transformation. `https://medium.com/analytics-vidhya/opencv-perspective-transformation-9edffefb2143`. Ημερομηνία πρόσβασης: 3-08-2021.

[17] Md Haque, Md Islam, Kazi Alam, Hasib Iqbal, and Md Shaik. A computer vision based lane detection approach. *International Journal of Image, Graphics and Signal Processing*, 11:27–34, 03 2019.

[18] Ziqiang Sun. Vision based lane detection for self-driving car. pages 635–638, 08 2020.

[19] Rakesh Dharoori and Sathiya Narayanan. *Vision-Based Lane Detection for Advanced Driver Assistance Systems*, pages 537–546. 01 2021.

[20] Shiuh-Jer Huang and Chien-Chang Tsai. Vehicle lane detection and following based on vision system and laser scanner. pages 1446–1449, 05 2017.

[21] Yang Xing, Chen Lv, Huaji Wang, Dongpu Cao, and Efstathios Velenis. Dynamic integration and online evaluation of vision-based lane detection algorithms. *IET Intelligent Transport Systems*, 13, 09 2018.

[22] Vitor Bottazzi, Paulo Borges, and Jun Jo. A vision-based lane detection system combining appearance segmentation and tracking of salient points. pages 443–448, 06 2013.

[23] King Hann Lim, Kah Seng, and Li-Minn Ang. Lane detection and kalman-based linear-parabolic lane tracking. *Intelligent Human-Machine Systems and Cybernetics, International Conference on*, 2:351–354, 01 2009.

[24] Weina Lu, Yucai Zheng, Yu Ma, and Tao Liu. An integrated approach to recognition of lane marking and road boundary. pages 649–653, 02 2008.

[25] Raspberry pi official page. `https://www.raspberrypi.org/`. Ημερομηνία πρόσβασης: 15-08-2021.

[26] Devobox. `https://www.devobox.com/en/motors/537-high-torque-mg995-servo-motor-metal-gear.html`. Ημερομηνία πρόσβασης: 09-09-2021.

[27] Grobotronics. `https://grobotronics.com`. Ημερομηνία πρόσβασης: 15-08-2021.

[28] Pinterest. `https://www.pinterest.com/`. Ημερομηνία πρόσβασης: 15-08-2021.

[29] Stackoverflow. `https://stackoverflow.com/questions/29331666/hsv-color-detection-with-opencv`. Ημερομηνία πρόσβασης: 18-08-2021.

[30] M. E. Moran. Evolution of robotic arms. *Journal of Robotic Surgery*, 1(2):103–111, 2007.

[31] Isaac Asimov. Runaround. *Astounding Science-Fiction*, March 1942.

[32] Johanna Wallén. The history of the industrial robot. Technical Report LiTH-ISY-R-2853, Department of Electrical Engineering, Linköpings University, SE-581 83 Linköping, Sweden, 2008.

[33] Y.S. Kwoh, J. Hou, E.A. Jonckheere, and S. Hayati. A robot with improved absolute positioning accuracy for ct guided stereotactic brain surgery. *IEEE Transactions on Biomedical Engineering*, 35(2):153–160, 1988.

[34] National Highway Traffic Safety Administration. 'national motor vehicle crash causation survey: report to congress'. Technical Report National Highway Traffic Safety Administration Technical Report DOT HS 811, July 2008.

[35] Tourtouras Evangelos. Computer vision with opencv c++/python. Technical report, Department of Imformatics and Telecommunication Engineering, University of Western Macedonia, 2018.

[36] Θ. Τσούκα Α. Αγγελίδου. Ανάπτυξη Λογισμικού Για Την Αναγνώριση Οπτικών Σημάτων Μέσω Βλεφαρίσματος(blink detection). Πτυχιακή εργασία, Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κεντρικής Μακεδονίας, 2013.

[37] Dr. Asha Jindal and Pratiksha Kadam. *Python Programming with Case Studies*. 05 2020.

[38] Physics4u. `https://www.physics4u.gr`. Ημερομηνία πρόσβασης: 27-07-2021.

[39] Β. Σ. Μπελέρης. Μηχανική Όραση. Πτυχιακή εργασία, Τμήμα Μηχανικών Αυτοματισμού, Οκτ. 2016.

[40] Estevao Gedraite and M. Hadad. Investigation on the effect of a gaussian blur in image filtering and segmentation. pages 393–396, 01 2011.

[41] Siddharth Misra and Yaokun Wu. Chapter 10 - machine learning assisted segmentation of scanning electron microscopy images of organic-rich shales with feature extraction and feature ranking. In Siddharth Misra, Hao Li, and Jiabo He, editors, *Machine Learning for Subsurface Characterization*, pages 289–314. Gulf Professional Publishing, 2020.

[42] D.R. Sona, C. Rajabai, D. Dey, A. Jain, R.R. Das, and Stephen Olabiyisi. A case study: Edge detection techniques using hough transform and canny edge algorithm. *International Journal of Mechanical Engineering and Technology*, 8:729–740, 10 2017.

[43] Towards data science. `https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549`. Ημερομηνία πρόσβασης: 2-08-2021.

[44] Tutorialspoint: Perspective transformation. `https://www.tutorialspoint.com/dip/perspective_transformation.htm`. Ημερομηνία πρόσβασης: 3-08-2021.

[45] Ardtech: Arduino technology. `https://ardtech.webnode.gr/l/pos-leitoyrgei-enas-servo-kinitiras/`. Ημερομηνία πρόσβασης: 15-08-2021.

[46] Σύνδεσμος google drive που περιέχει τον κώδικα και τα βίντεο της παρουσίασης του οχήματος. `https://drive.google.com/drive/folders/1HwsPsMS7gOCQY6dLb-aU-TO0rgkG8mSc?usp=sharing`. Ημερομηνία πρόσβασης: 15-01-2022.

ΠΑΡΑΡΤΗΜΑΤΑ

Παράρτημα Α

Αναγνώριση δύο κόκκινων γραμμών με λειτουργία αποτροπής γρήγορης στροφής

```
# -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
# -- mokkasmich@gmail.com

# -- Εισαγωγή Βιβλιοθηκών
import cv2
import numpy as np
import math
import MotorModule
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
# -- Ορισμός κάποιων σταθερών για τη εμφάνιση επεξερασεμένης εικόνας,
# -- λειτουργία των κινητήρων κλπ
_SHOW_IMAGE = 1
SHOW_MASK=0
MOTORS_ON=1
TURN=0

# -- Η συνάρτησης αρχικής επεξεργασίας της εικόνας
# -- που επιστρέφει τις ακμές
def detect_edges(frame):
    # -- Εφαρμογή φίλτρου θολώματος Gauss
    blur = cv2.GaussianBlur(frame, (5, 5), 0)
    # -- Μετατροπή της εικόνας από RGB σε HSV
```

```
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
    # -- Δημιουργία μάσκας για μόνο τις κόκκινες αποχρώσεις
    lower_red = np.array([0, 50, 20])
    upper_red = np.array([20, 255, 255])
    mask1 = cv2.inRange(hsv, lower_red, upper_red)
    lower_red = np.array([140, 50, 20])
    upper_red = np.array([180, 255, 255])
    mask2 = cv2.inRange(hsv,lower_red,upper_red)
    mask=mask1+mask2
    if SHOW_MASK:
        cv2.imshow("Mask",mask)
    # -- Εφαρμογή αλγορίθμου Canny για εύρεση ακμών
    edges = cv2.Canny(mask, 100, 200)
    return edges
# -- Συνάρτηση που δημιουργεί νέα εικόνα που περιέχει μόνο
# -- ένα πλαίσιο της αρχικής εικόναςς
def select_region_of_interest(edges):

    height, width = edges.shape
    mask = np.zeros_like(edges)
    # -- Εφαρμογή του πλαισίου από τις δύο κάτω γωνίες
    # -- μέχρι 2/5 του συνολικού ύψους της εικόνας προς τα επάνω
    area = np.array([[(0, height * 3/5), (width, height * 3/5), (width,
   height), (0, height),]], np.int32)
    # -- Αντιγραφή της πληροφορίας που βρίσκεται μέσα
    # -- στο πλαίσιο στη νέα εικόνα
    cv2.fillPoly(mask, area, 255)
    cropped_edges = cv2.bitwise_and(edges, mask)

    return cropped_image

# -- Συνάρτηση αναγνώρισης γραμμών στην εικόνα
def line_detection(cropped_edges):
    rho = 1
    angle = np.pi / 180
    min_threshold = 10 # Κατώτατος αριθμός ψήφων για την νέα γραμμή
    minLineLength=30 # Ελάχιστος αριθμός σημείων που
                     # απαρτίζουν την γραμμή
    maxLineGap=5 # Ελάχιστος αριθμός απόστασης μεταξύ
```

```
                # δύο διαδοχικών σημείων

    # -- Εφαρμογή Μετασχηματισμού Hough και αναγνώριση γραμμών
    # -- και αποθήκευση των συντεταγμένων
    # -- των ακραίων σημείων τους σε μια λίστα
    lines_detected = cv2.HoughLinesP(cropped_edges, rho, angle,
    min_threshold,np.array([]), minLineLength, maxLineGap)

    return lines_detected




# -- Συνάρτηση η οποία διαλέγει ποιες γραμμές ανοίκουν
# -- στην αριστερή γραμμή και ποιες στην δεξιά,
# -- συγκρίνοντας την κλίση των γραμμών μέσα στην λίστα

def slope_classification(frame, line_segments):

    # -- Έλεγχος αν η λίστα που έχει δοθεί είναι άδεια
    # -- σημαίνει πως δεν έχουν αναγνωριστεί γραμμές
    lane_lines = []
    if line_segments is None:
        return lane_lines

    height, width, _ = frame.shape

    # -- Δημιουργία δύο λιστών που περιέχουν τις γραμμές που ανήκουν
    # -- στη δεξιά και αριστερή γραμμή της λωρίδας
    left_fit = []
    right_fit = []

    boundary = 1/3
    # -- Καθορισμός ορίου: η δεξιά γραμμή πρέπει να βρίσκεται
    # -- στα δεξιά 2/3 της εικόνας
    # -- ενώ η αριστερή στα αριστερά 2/3
    left_region_boundary = width * (1 - boundary)
```

```
    right_region_boundary = width* boundary

    # -- Επανάληψη για κάθε γραμμή της λίστας
    for line_segment in line_segments:
        for x1, y1, x2, y2 in line_segment:

            # -- Αποφυγή κάθετης γραμμής γιατί θα προκύψει απειρο
            if x1 == x2:
                continue
            # -- Υπολογισμός πολυωνύμου πρώτου βαθμού
            # -- της αναγνωρισμένης γραμμής
            fit = np.polyfit((x1, x2), (y1, y2), 1)

            # -- Η κλίση της γραμμής
            slope = fit[0]
            # -- Το σημείο που τέμνει τον κατακόρυφο άξονα
            # -- η προέκταση της γραμμής
            intercept = fit[1]

            # -- Αν η κλίση αρνητική, τότε ανήκει
            # -- στην αριστερή γραμμή της λωρίδας
            if slope < 0:
                if x1 < left_region_boundary and x2 <
    left_region_boundary:
                    left_fit.append((slope, intercept))
            # -- Αν η κλίση θετική, τότε ανήκει
            # -- στην δεξιά γραμμή της λωρίδας
            else:
                if x1 > right_region_boundary and x2 >
    right_region_boundary:
                    right_fit.append((slope, intercept))

    # -- Υπολογισμός της μέσης γραμμής από όλες τις γραμμές
    # -- που ανήκουν στην αριστερή γραμμή της λωρίδας
    left_fit_average = np.average(left_fit, axis=0)
    if len(left_fit) > 0:
        lane_lines.append(make_points(frame, left_fit_average))

    # -- Υπολογισμός της μέσης γραμμής από όλες τις γραμμές
```

```
138     # -- που ανήκουν στην δεξιά γραμμή της λωρίδας
139     right_fit_average = np.average(right_fit, axis=0)
140     if len(right_fit) > 0:
141         lane_lines.append(make_points(frame, right_fit_average))
142 
143     return lane_lines
144 
145 
146 
147 
148 
149 
150 
151 
152 # -- Συνάρτηση που χρησιμοποιείται απο πάνω για την εξαγωγή
153 # -- των ακραίων σημείων της τελικής αναπαράστασης
154 # -- της δεξιάς και αριστερής γραμμής
155 def make_points(frame, line):
156     height, width, _ = frame.shape
157     slope, intercept = line
158     # -- Τα x και y πρέπει να ανήκουν μέσα στα όρια της εικόνας
159     # -- για να γίνει ο σχεδιασμός τους
160     y1 = height
161     y2 = int(y1 * 1 / 2) # Ύψος μέχρι το μέσο της εικόνας
162     x1 = max(-width, min(2 * width, int((y1 - intercept) / slope)))
163     x2 = max(-width, min(2 * width, int((y2 - intercept) / slope)))
164     return [[x1, y1, x2, y2]]
165 
166 # -- Συνάρτηση για την αναπαράσταση των γραμμών πάνω στο frame
167 def display_lines(frame, lines, line_color=(0, 255, 0), line_width=2):
168     line_image = np.zeros_like(frame)
169 
170     # -- Αποφυγή αναγνώρισης μιας γραμμής λόγω τους πάχους σαν δύο,
171     # -- ελέγχοντας την απόσταση και τη θέση των γραμμών,
172     # -- και διατήρηση μόνο της μίας από τις δύο,
173     # -- ανάλογα με την τοποθεσία της
174     # -- Τέλος ακολουθεί ο σχεδιασμός των γραμμών που επιτράπηκαν
175     if len(lines)==2:
176         [[left_x1,left_y1,left_x2,left_y2]] = lines[0]
```

```
177         [[right_x1,right_y1,right_x2,right_y2]] = lines[1]
178         if abs(left_x2-right_x2)<50:
179             if left_x1<220:
180                 lines.pop(1)
181             else:
182                 lines.pop(0)
183             [[x1,y1,x2,y2]] = lines[0]
184             cv2.line(line_image, (x1,y1), (x2, y2), line_color,
    line_width)
185         else:
186             cv2.line(line_image, (left_x1, left_y1), (left_x2, left_y2),
    line_color, line_width)
187             cv2.line(line_image, (right_x1, right_y1), (right_x2,
    right_y2), line_color, line_width)
188     # -- Περίπτωση Αναγνώρισης μίας μόνο γραμμής
189     elif len(lines)==1:
190         [[x1,y1,x2,y2]] = lines[0]
191         cv2.line(line_image, (x1,y1), (x2, y2), line_color, line_width)
192 
193     line_image = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
194     return line_image
195 
196 
197 # -- Συνάρτηση υπολογισμού γραμμής πορείας του οχήματος
198 # -- με βάση την κλίση και των συντεταγμένων των γραμμών της λωρίδας
199 def compute_steering_angle(frame, lane_lines):
200     # -- Περίπτωση καμίας γραμμής
201     if len(lane_lines) == 0:
202         return -90
203     height, width, _ = frame.shape
204 
205     # -- Περίπτωση αναγνώρισης μίας γραμμής
206     if len(lane_lines) == 1:
207         logging.debug('Only detected one lane line, just follow it. %s' %
     lane_lines[0])
208         x1, _, x2, _ = lane_lines[0][0]
209         x_offset = x2 - x1
210 
211     # -- Περίπτωση αναγνώρισης δύο γραμμών
```

```
212     else:
213         _, _, left_x2, _ = lane_lines[0][0]
214         _, _, right_x2, _ = lane_lines[1][0]
215         camera_offset = 0.02 # -- Κανονικοποίηση για κεντράρισμα
216         mid = int(width / 2 * (1 + camera_offset))
217         x_offset = (left_x2 + right_x2) / 2 - mid
218 
219     # -- Υπολογισμός γραμμής πορείας του οχήματος στο μέσο
220     # -- των δύο γραμμών της λωρίδας
221 
222     y_offset = int(height / 2)
223 
224     angle_to_mid_radian = math.atan(x_offset / y_offset)
225 
226     # -- Μετατροπή από ακτίνια σε μοίρες
227     angle_to_mid_deg = int(angle_to_mid_radian * 180.0 / math.pi)
228     steering_angle = angle_to_mid_deg + 90
229 
230     return steering_angle
231 
232 # -- Συνάρτηση σχεδιασμού της γραμμής πορείας στο frame,
233 # -- που αρχίζει από το μέσο του κάτω μέρους του frame
234 # -- και έχει ίδια κλίση με την πορεία του οχήματος
235 def display_heading_line(frame, steering_angle, line_color=(0, 0, 255),
        line_width=5 ):
236     heading_image = np.zeros_like(frame)
237     height, width, _ = frame.shape
238 
239     steering_angle_radian = steering_angle / 180.0 * math.pi
240     x1 = int(width / 2)
241     y1 = height
242     x2 = int(x1 - height / 2 / math.tan(steering_angle_radian))
243     y2 = int(height / 2)
244 
245     cv2.line(heading_image, (x1, y1), (x2, y2), line_color, line_width)
246     heading_image = cv2.addWeighted(frame, 0.8, heading_image, 1, 1)
247 
248     return heading_image
249 
```

```
# -- Συνάρτηση που υπολογίζει και σχεδιάζει την γραμμή πορείας
# -- του οχήματος καλώντας τις δύο παραπάνω συναρτήσεις
def steer(frame, lane_lines):

        # -- Περίπτωση καμίας αναγνωρισμένης γραμμής
        if len(lane_lines) == 0:
            return frame

        new_steering_angle = compute_steering_angle(frame, lane_lines)
        curr_heading_image = display_heading_line(frame,
    new_steering_angle)

        return curr_heading_image, new_steering_angle


# -- Συνάρτηση για την εμφάνιση ή όχι της ζωντανής εικόνας του οχήματος
def show_image(title, frame, show=_SHOW_IMAGE):
    if show:
        cv2.imshow(title, frame)

####       MAIN           ######
if __name__ == "__main__":
    motor = MotorModule.Motor(4,3,2,22,17,27)
    try:
    #-- Αρχικοποίηση ρυθμίσεων κάμερας
        camera = PiCamera()
        camera.resolution = (320, 240)
        camera.brightness = 55
        camera.framerate = 32
        rawCapture = PiRGBArray(camera, size=(320, 240))
        time.sleep(0.5)
        count=0

    #-- Το κύριο loop το οποίο παίρνει ένα ένα τα frames για την
    επεξεργασία
        while True:

            #-- Σύλληψη του frame από την κάμερα σε κωδικοποίηση RGB
            camera.capture(rawCapture, format="bgr", use_video_port=True)
```

```
287             frame = rawCapture.array
288             timer = cv2.getTickCount() #Μετρητής-- για υπολογισμό frames
    ανά δευτερόλεπτο
289
290             # -- Κλήση της συνάρτησης που επεξεργάζεται το frame
291             # -- και επιστρέφει μια εικόνα με μόνο τις ακμές
292             edges = image_process(frame)
293
294             # -- Ορισμός πλαισίου που θα γίνεται η ανίχνεσυη λωρίδας
295             cropped_edges = select_region_of_interest(edges)
296
297             # -- Ανίχνευση για γραμμές
298             line_segments = line_detection(cropped_edges)
299
300             # -- Κατανομή των γραμμών σε δεξιά και αριστερή
301             lane_lines = slope_classification(frame, line_segments)
302
303             # -- Δημιουργία frame με τις ανιχνευμένες γραμμές
304             lane_lines_image = display_lines(frame, lane_lines)
305
306             # -- Περίπτωση που ανιχνεύεται μία γραμμή να μη στρίβει
307             # -- πολύ νωρίς σε περίπτωση απότομης κλίσης
308             if len(lane_lines)==1:
309                 for line in lane_lines:
310                     [[x1, y1, x2, y2]]=line
311                     if x1<-20 or x1>320:
312                         print("HIGH")
313                         TURN=0
314                     else:
315                         TURN=1
316
317             # -- Περίπτωση που ανιχνεύονται γραμμές στο frame
318             if len(lane_lines)>0:
319
320                 # -- Υπολογιμός πορείας οχήματος με βάση τη κλίση της
    λωρίδας
321                 combo_image, new_steering_angle = steer(frame, lane_lines
    )
322
```

```
323                 # -- Εκτύπωση στο frame την κλίση
324                 cv2.putText(combo_image, str(new_steering_angle), (50,50)
    , cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 1, cv2.LINE_AA)
325 
326                 # -- Εντολές για την κίνηση των τροχών αναλόγως την
    περίπτωση
327                 if MOTORS_ON:
328                     if new_steering_angle > 160 and TURN:
329                         motor.move(0.1, -0.4,  0)#0.1 , -0.4
330                         count=0
331                         print('Steep Right')
332                     elif new_steering_angle > 110and TURN:
333                         motor.move(0.1, -0.25,  0)
334                         count=0
335                         print('Right')
336                     elif new_steering_angle < 20 and TURN:
337                         count=0
338                         motor.move(0.1, 0.4, 0) #0.1 , 0,4
339                         print('Steep Left')
340                     elif new_steering_angle < 70 and TURN:
341                         count=0
342                         motor.move(0.1, 0.25, 0)
343                         print('Left')
344                     else:
345                         count=0
346                         motor.move(0.15, 0, 0)
347 
348             else:
349             # -- Σε περίπτωση μη αναγνώρισης καμίας γραμμής για κάποιο
    χρόνο,
350             # -- οπισθέν για να ξαναμπεί στη λεωρίδα εφόσον βγήκε
351                 count=count+1
352                 if MOTORS_ON:
353                     if count==15:
354                         motor.move(-0.3,0,0.2)
355                         count=0
356                 motor.stop(0)
357                 combo_image = steer(frame, lane_lines)
358 
```

```
359             if _SHOW_IMAGE:
360             # -- Εμφάνιση της εικόνας του frame με τις ανιχνευμένες
    γραμμές,
361             # -- την κλίση της πορείας και τα fps
362                 fps = cv2.getTickFrequency() / (cv2.getTickCount() -
    timer);
363                 cv2.putText(lane_lines_image, "FPS : " + str(int(fps)),
    (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);
364
365                 resized = cv2.resize(lane_lines_image,(640,480))
366                 cv2.imshow("Road with Lane line", resized)
367
368     #Προετοιμασία για να πάρει το νεότερο frame στο επόμενο loop
369
370             rawCapture.truncate(0)
371     #Έλεγχος λήξης προγράμματος άμα πατηθεί το q
372             if cv2.waitKey(1) & 0xFF == ord('q'):
373                 motor.stop(0)
374                 break
375     finally:
376         cv2.destroyAllWindows()
```

Παράρτημα Β

Αλγόριθμος ακολουθίας μίας γραμμής

Β.1 Ακολουθία μίας γραμμής

Οι βιβλιοθήκες που χρησιμοποιήθηκαν είναι:

1. cv2 (Η βιβλιοθήκη της OpenCV)

2. numpy (Numeric Python: Μια βιβλιοθήκη που επιτρέπει την αναπράσταση διαφόρων λιστών πολλών διαστάσεων και εύκολη πραγματοποίηση πολλών πράξεων σε αυτές)

3. time.sleep (Η συνάρτηση sleep, η οποία διακόπτει την διεργασία του προγράμματος για όσο χρόνο έχει οριστεί)

4. math (Math Library: Η βιβλιοθήκη που δίνει εύκολη πρόσβαση σε ορισμένες μαθηματικές συναρτήσεις)

5. MotorCode(Το αρχείο κώδικα για τον έλεγχο των τροχών κίνησης του οχήματος)

Β.1.1 Περιγραφή αλγορίθμου αναγνώρισης μιας γραμμής

Algorithm 2 Αλγόριθμος Αναγνώρισης Μιας Γραμμής

while 1 **do**

Διάβασε νέο frame από την κάμερα

Μετατροπή μεγέθους frame σε μικρότερη ανάλυση

Εφαρμογή φίλτρου θολώματος Gauss (Gaussian Blur)

Μετατροπή κωδικοποίησης εικόνας από RGB σε HSV

Δημιουργία νέας εικόνας που περιέχει μόνο τις κόκκινες αποχρώσεις της εικόνας

Εφαρμογή φίλτρου Canny για την εύρεση ακμών

Διατήρηση πλαισίου στο κάτω μέρος της εικόνας (Region OF Interest)

Αναγνώριση κομματιών της γραμμής

Εύρεση της τελικής γαμμής από τον συνδυασμό των ανιχνευμένων κομματιών

Εύρεση της γωνίας κίνησης του οχήματος με βάση την κλίση και την θέση της ανιχνευμένης γραμμής

Εμφάνιση της γραμμής πάνω στο τελικό frame

Οδήγηση του οχήματος

end while

Β.1.2 Detect Edges

Είναι ίδια με αυτήν του αλγορίθμου αναγνώρισης διπλής λωρίδας. (Κώδ. Β.1)

```
327
328 # -- Μόκκας Μιχαήλ - Διπλωματική Εργασία
329 # -- mokkasmich@gmail.com
330
331 def detect_edges(frame):
332     # -- Εφαρμογή φίλτρου θολώματος Gauss
333     blur = cv2.GaussianBlur(frame, (5, 5), 0)
334     # -- Μετατροπή της εικόνας από RGB σε HSV
335     hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
336     # -- Δημιουργία μάσκας για μόνο τις κόκκινες αποχρώσεις
337     lower_red = np.array([0, 50, 20])
338     upper_red = np.array([20, 255, 255])
339     mask1 = cv2.inRange(hsv, lower_red, upper_red)
```

```
340     lower_red = np.array([140, 50, 20])
341     upper_red = np.array([180, 255, 255])
342     mask2 = cv2.inRange(hsv,lower_red,upper_red)
343     mask=mask1+mask2
344     if SHOW_MASK:
345         cv2.imshow("Mask",mask)
346     # -- Εφαρμογή αλγορίθμου Canny για εύρεση ακμών
347     edges = cv2.Canny(mask, 100, 200)
348     return edges
```

Κώδικας B.1: Συνάρτηση ανίχνευσης των ακμών στην εικόνα

B.1.3 Region of Interest

Η συνάρτηση Select Region of Interest (Κώδ. B.2) παίρνει σαν όρισμα το frame με τις ακμές, και επιλέγει σαν πλαίσιο ενδιαφέροντος το κατω μισό κομμάτι του frame, με στόχο τον εντοπισμό των γραμμών μόνο στο κάτω μέρος της εικόνας όπου εκεί αναμένεται να είναι και η γραμμή του δρόμου. Με αυτον τον τρόπο, επίσης, δεν θα γίνει λανθασμένη αναγνώριση από άλλα αντικείμενα κόκκινου χρώματος στον χώρο, ενώ επιταχύνεται και η επεξεργασία του κάθε frame, καθώς η αναγνώριση γραμμών με χρήση του μετασχηματισμού Hough θα γίνεται μόνο στο μισό frame κάθε φορά.

```
349 # -- Συνάρτηση που επιστρέφει μια εικόνα η οποία
350 # -- περιέχει μόνο τις πληροφορίες που βρίσκονται
351 # -- σε ένα συγκεκριμένο πλαίσιο της εικόνας
352 def select_region_of_interest(image):
353     height = image.shape[0]
354     # -- Διατήρηση του κατώτερου μισού frame
355     polygons = np.array([[(0,480),(640,480),(640, 240), (0,240)]])
356     mask=np.zeros_like(image)
357     cv2.fillPoly(mask, polygons, 255)
358     masked_image=cv2.bitwise_and(image,mask)
359     return masked_image
```

Κώδικας B.2: Συνάρτηση επιλογής πλαισίου ενδιαφέροντος

Β.1.4 Average Slope

Η συνάρτηση Average Slope (Κώδ. Β.3) παίρνει σαν ορίσματα το frame και τις αναγνωρισμένες γραμμές έπειτα από την εφαρμογή του αλγορίθμου του Hough. Βρίσκει τις κλίσεις του κάθε κομματιού γραμμής που αναγνωρίστηκε και στην συνέχεια υπολογίζει τον μέσο όρο, που θα είναι η τελική τιμή της κλίσης της γραμμής.

```
360 # -- Υπολογισμός της μέσης κλίσης δοσμένων των γραμμών που αναγνωρίστηκαν
361 def average_slope(image,lines):
362     fit = []
363     if lines is not None:
364         for line in lines:
365             x1,y1,x2,y2 = line.reshape(4)
366             parameters = np.polyfit((x1,x2),(y1,y2),1)
367             slope = parameters[0]
368             intercept = parameters[1]
369             fit.append((slope, intercept))
370         # -- Υπολογισμός μέσης τιμής των γραμμών
371         if fit :
372             fit_avrg = np.average(fit, axis=0)
373             slope,_=fit_avrg
374             line = make_coords(image, fit_avrg)
375 
376             return line,slope
```

Κώδικας Β.3: Συνάρτηση εύρεσης της τελικής κλίσης της γραμμής

Β.1.5 Display Lines

Έπειτα, ο αλγόριθμος με τη χρήση της συνάρτησης Display Lines (Κώδ. Β.4) θα αναπαραστήσει την ανιχνευμένη γραμμή στο frame, με στόχο τον έλεγχο της σωστής λειτουργίας του αλγορίθμου.

```
377 # -- Συνάρτηση για την εμφάνιση πάνω στο frame
378 # -- της γραμμής που ανιχνεύθηκε
379 def displayLines(image, lines):
380     line_image = np.zeros_like(image)
381     if lines is not None:
382         x1,y1,x2,y2 = lines.reshape(4)
```

```
383         cv2.line(line_image,(round(x1,3),round(y1,3)),(round(x2,3),round(
    y2,3)), 255, 20)
384     return line_image
```

Κώδικας B.4: Συνάρτηση απεικόνησης της ανιχνευμένης γραμμής πάνω στο frame

B.1.6 Κύρια Συνάρτηση

Τέλος, καλείται η Κύρια Συνάρτηση του προγράμματος, όπου επιτελεί αρχικοποίηση της λειτουργίας της κάμερας και των κινητήρων του οχήματος. Έπειτα, ξεκινάει το loop το οποίο περιλαμβάνει τον κώδικα που εκτελείται για κάθε frame της κάμερας, μέχρι την διακοπή εκτέλεσης του προγράμματος. Μετά την επιστροφή από την συνάρτηση επιλογής πλαισίου ενδιαφέροντος, εφαρμόζεται στη γραμμή κώδικα 394 η συνάρτηση της OpenCV για την εύρεση τμημάτων γραμμών με το μετασχηματισμό του Hough. Από τα τμήματα που προέκυψαν λαμβάνεται η μέση τιμή των κλίσεων και της τοποθεσίας των γραμμών και εξάγεται η τελική ανιχνευμένη γραμμή. Με την κλήση της συνάρτησης Display Lines σχηματίζεται η γραμμή αυτή πάνω στο frame. Στη συνέχεια, επιτελείται η απόφαση για την κίνηση των τροχών αναλόγως πρώτα την θέση της γραμμής και έπειτα την κλίση της. Δηλαδή αν το κέντρο της γραμμής είναι κοντά στα άκρα τότε το όχημα στρίβει προς εκείνη την κατεύθυνση ώστε να κεντραριστεί και στη συνέχεια εφόσον η γραμμή είναι σε κεντρικό σημείο γίνεται ο έλεγχος της κλίσης της γραμμής. Αν για παράδειγμα είναι απότομη η κλίση προς τα αριστερά τότε το όχημα θα στρίψει εντονότερα προς τα αριστερά, ενώ αν η κλίση είναι μικρή θα στρίψει ομαλότερα. Τέλος, αν η κλίση είναι αμηλητέα το όχημα θα κινηθεί ευθεία.

```
385 ### MAIN ###
386
387 camera = PiCamera()
388 camera.resolution = (320, 240)
389 camera.brightness = 55
390 camera.framerate = 32
391 rawCapture = PiRGBArray(camera, size=(320, 240))
392 time.sleep(0.5)
393
394 motor= MotorModule.Motor(4,3,2,22,17,27)
395
396 while (1):
397     # -- Σύλληψη του νεότερου frame από την κάμερα σε κωδικοποίηση RGB
```

```
398     camera.capture(rawCapture, format="bgr", use_video_port=True)
399     image = rawCapture.array
400 
401     # -- Κλήση της συνάρτησης για επεξεργασία της εικόνας
402     # -- και επιστρέφει μια εικόνα με τις ακμές
403     processed_image = detect_edges(image)
404 
405     # -- Ορισμός πλαισίου που θα γίνεται η ανίχνευση της γραμμής
406     cropped_image = select_region_of_interest(processed_image)
407 
408     lane_image=np.copy(image)
409 
410     # -- Εφαρμογή του αλγορίθμου του Hough για αναγνώριση κομματιών από
    γραμμές πάνω στην εικόνα ακμών και αποθήκευσή τους σε μια λίστα
411     lines=cv2.HoughLinesP(cropped_image,2,np.pi/180,100,np.array([]),
    minLineLength=40,maxLineGap=100)
412 
413     # -- Σε περίπτωση ανίχνευσης γραμμών:
414     if lines is not None:
415         # -- Υπολογισμός μέσης κλίσης των ανιχνευμένων γραμμών
416         avg_lines , slope = average_slope(lane_image, lines)
417         x1,y1,x2,y2=avg_lines
418 
419         # -- Υπολογισμός γωνίας της τελικής γραμμής
420         theta=abs(math.degrees(math.atan2((y2-y1),(x2-x1))))
421 
422         # -- Υπολογισμός των μέσων σημείων των γραμμών
423         y_center_of_line = int((y2+y1)/2)
424         x_center_of_line = int((x2+x1)/2)
425 
426         # -- Κλήση της συνάρτησης για εμφάνιση των γραμμών πάνω στο frame
427         line_image=displayLines(lane_image,avg_lines)
428         final_image=cv2.addWeighted(lane_image,0.8,line_image,1.1,1)
429 
430         # -- Εντολές για την κίνηση των τροχών αναλόγως την θέση
431         # -- και την κλίση της γραμμής
432         if x_center_of_line<110:
433             motor.move(0.2, 0.2, 0.05)
434             motor.stop(0)
```

```
435         elif x_center_of_line>490:
436             motor.move(0.2, -0.2, 0.05)
437             motor.stop(0)
438         # -- Περιπτώσεις οι οποίες λύθηκαν με την χρήση
439         # -- της νέας ευρυγώνειας κάμερας
440 # --         elif theta<45 and x_center_of_line<200:
441 # --             motor.move(0.2, 0, 0.05)
442 # --             motor.stop(0)
443 # --             motor.move(0.2, -0.5, 0.05)
444 # --             motor.stop(0)
445 # --         elif theta>150 and x_center_of_line>400:
446 # --             motor.move(0.2, 0, 0.05)
447 # --             motor.stop(0)
448 # --             motor.move(0.2, 0.5, 0.05)
449 # --             motor.stop(0)
450         elif theta>110:
451             motor.move(0.2, 0.2, 0.05)
452             motor.stop(0)
453             cap.read()
454         elif theta<80:
455             motor.move(0.2, -0.2, 0.05)
456             motor.stop(0)
457             cap.read()
458         else:
459             motor.move(0.2, 0, 0.05)
460             cap.read()
461             motor.stop(0)
462     # -- Προετοιμασία ώστε το επόμενο frame να είναι το νεότερο
463     rawCapture.truncate(0)
464 
465     # -- Τερματισμός Λειτουργίας με το πάτημα του q
466     key = cv2.waitKey(1)
467     if key== ord('q'):
468         break
469 
470 cv2.destroyAllWindows()
471 motor.stop(0)
```

Κώδικας B.5: Κύρια συνάρτηση ακολουθίας μίας γραμμής